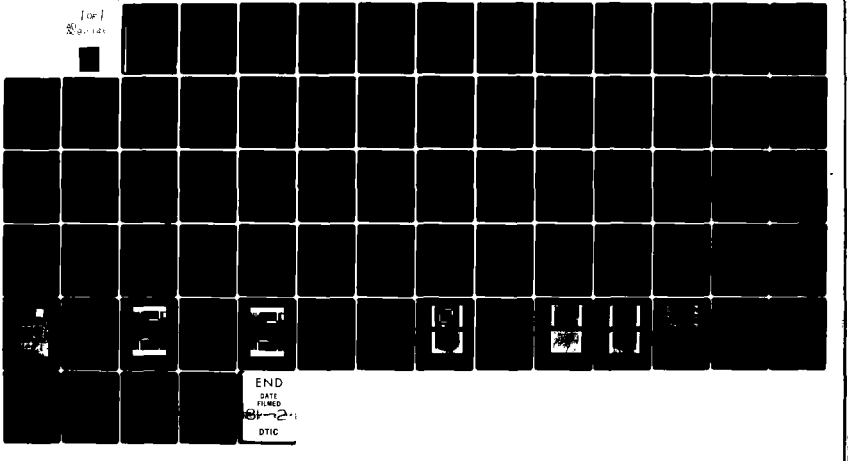
AD-A092 146 SRI INTERNATIONAL MENLO PARK CA F/G 9/2
VIDEO STREAM PROCESSORS: A COST-EFFECTIVE COMPUTATIONAL ARCHITE--ETC(U)
JUN 80 J M TENENBAUM DAAK70-78-C-0114
UNCLASSIFIED ETL-0229 NL



ETL-0229

VIDEO STREAM PROCESSORS: A COST-EFFECTIVE COMPUTATIONAL ARCHITECTURE FOR IMAGE PROCESSING

June 1980

Final Report covering the period
22 September 1978 to 21 November 1979

Jay M. Tenenbaum

SRI International
333 Ravenswood Avenue
Menlo Park, California 94025

Prepared for:

U.S. Army Engineer Topographic Laboratories
Fort Belvoir, Virginia 22060

Approved for public release; distribution unlimited.

NOTICES

Destroy this report when no longer needed.
Do not return it to the originator.

The findings in this report are not to be construed as an official Department of the Army position unless so designated by other authorized documents.

The citation in this report of trade names of commercially available products does not constitute official endorsement or approval of the use of such products.

UNCLASSIFIED (9) Final Rept. 22 Sep 78—21 Nov 79.

SECURITY CLASSIFICATION OF THIS PAGE (When Data Entered)

(18) (19) **REPORT DOCUMENTATION PAGE**

READ INSTRUCTIONS BEFORE COMPLETING FORM

1. REPORT NUMBER	2. GOVT ACCESSION NO.	3. RECIPIENT'S CATALOG NUMBER
ETL-0229	AD-A092 146	

4. TITLE (and Subtitle)

VIDEO STREAM PROCESSORS: A COST-EFFECTIVE COMPUTATIONAL ARCHITECTURE FOR IMAGE PROCESSING

5. TYPE OF REPORT & PERIOD COVERED

Final Report
9-22-78 to 11-21-79

6. PERFORMING ORG. REPORT NUMBER

SRI Project 7864

(10) 7. AUTHOR(s)

Jay M. Tenenbaum

8. CONTRACT OR GRANT NUMBER(s)

(15) Contract No.
DAAK70-78-C-0114

9. PERFORMING ORGANIZATION NAME AND ADDRESS

SRI International
333 Ravenswood Avenue
Menlo Park, CA 94025

10. PROGRAM ELEMENT, PROJECT, TASK AREA & WORK UNIT NUMBERS

(11)

11. CONTROLLING OFFICE NAME AND ADDRESS

U.S. Army Engineer Topographic Laboratories
Fort Belvoir, Virginia 22060

12. REPORT DATE: June 1980

13. NO. OF PAGES: vi+63

15. SECURITY CLASS. (of this report)

UNCLASSIFIED

14. MONITORING AGENCY NAME & ADDRESS (If diff. from Controlling Office)

(12) 72

15a. DECLASSIFICATION/DOWNGRADING SCHEDULE

16. DISTRIBUTION STATEMENT (of this report)

Approved for public release; distribution unlimited.

17. DISTRIBUTION STATEMENT (of the abstract entered in Block 20, if different from report)

18. SUPPLEMENTARY NOTES

19. KEY WORDS (Continue on reverse side if necessary and identify by block number)

Display Processor, Image Processing, Video Stream Processor

20. ABSTRACT (Continue on reverse side if necessary and identify by block number)

This report evaluates the capabilities of a new class of image-processing systems, known generically as video stream processors (VSPs). VSPs are an outgrowth of image display technology; image data from the display memory are streamed at video rates through a digital-processing unit and back to memory for subsequent display or further processing. This architecture serially simulates a parallel-array processor and is capable, in principle, of executing any locally parallel operation, such as convolution and edge detection, at a fraction of the cost of a truly parallel system. (cont.)

DD FORM 1 JAN 73 1473
EDITION OF 1 NOV 65 IS OBSOLETE

UNCLASSIFIED
SECURITY CLASSIFICATION OF THIS PAGE (When Data Entered)

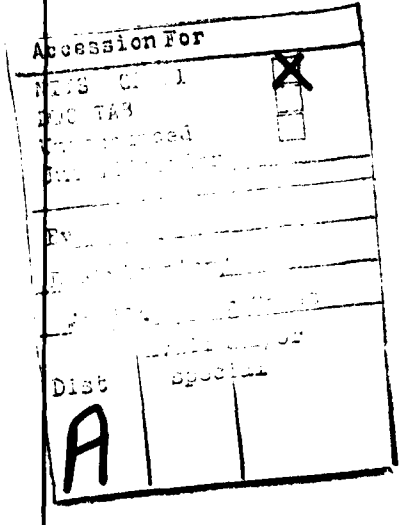
410281

19. KEY WORDS (Continued)

20 ABSTRACT (Continued)

Our evaluation begins with a general discussion of the architecture and use of VSPs that highlights the fundamental concepts and vast application potential of this class of machines. This discussion is based on a hypothetical VSP design in order to avoid artificial constraints imposed by design limitations of any particular commercial product. The hypothetical design also serves as a standard against which current implementations can be evaluated.

The report next summarizes our experience with the IP-5000 Image Array Processor ~~(manufactured~~ by De ~~Anza Systems~~, Santa Clara, Cali~~fornia),~~ currently the most advanced commercially available VSP. The IP-5000 design is critiqued in the context of the hypothetical design, followed by a presentation of experimental results at SRI. The concluding discussion analyzes the IP-5000's limitations and proposes design refinements that would significantly improve its utility.



CONTENTS

ILLUSTRATIONS

TABLES

PREFACE

Professor Ed Riseman (University of Massachusetts) and David Smith (Carnegie-Mellon University) made substantial contributions in the formulative stages of this study.

This report was prepared for the U. S. Army Engineer Topographic Laboratories, Fort Belvoir, Virginia, under Contract No. DAAK70-78-C-0114. The Contracting Officer's Representative was Mr. F. Raye Norvelle.

I INTRODUCTION

This report evaluates the capabilities of a new class of image-processing systems known generically as video stream processors (VSPs). VSPs are an outgrowth of image display technology; image data from the display memory are streamed at video rates through a digital-processing unit and back to memory for subsequent display or further processing. This architecture serially simulates a parallel-array processor and is capable, in principle, of executing any locally parallel operation, such as convolution and edge detection, at a fraction of the cost of a truly parallel system. Moreover, VSP throughput can match that of current generation parallel processors such as Staran, whose performance is limited by I/O bottlenecks in windowing images from a secondary store.

Our evaluation begins with a general discussion of the architecture and use of VSPs that highlights the fundamental concepts and vast application potential of this class of machines. This discussion is based on a hypothetical VSP design in order to avoid artificial constraints imposed by design limitations of any particular commercial product. The hypothetical design also serves as a standard against which current implementations can be evaluated.

The report next summarizes our experience with the IP-5000 Image Array Processor (manufactured by De Anza Systems, Santa Clara, California), currently the most advanced commercially available VSP. The IP-5000 design is critiqued in the context of the hypothetical design, followed by a presentation of experimental results at SRI. In summary, the IP-5000 is capable of performing many low-level operations one to two orders of magnitude faster than conventional serial processors, including arithmetic and logical operations on images, real-time photometric correction of video imagery, local neighborhood operations (such as edge detection and unsharp masking), and convolution

and correlation. However, the primitiveness of the processor and the limited storage available for temporary results make programming very difficult. Also, the IP-5000's memory architecture effectively precludes propagation algorithms which are important for distance transforms, skeletonization, and the like.

The concluding discussion analyzes these limitations and proposes design refinements that would significantly improve the IP-5000's utility.

II BACKGROUND

The need for high-speed, low-cost computation is ubiquitous in image processing, and especially so in cartographic applications where both volume and resolution of imagery are typically very high.

Three types of computer organizations are currently used for high-volume image-processing applications:

(1) Large, general-purpose serial processors, such as the CDC 7600

(2) Parallel-array processors, such as the Goodyear Staran

(3) Video stream processors, such as the Comtal 8000 series, Stanford Technology Corporation Model 70/E, and De Anza Systems Model IP-5000.

The fastest available general-purpose processors can execute about 100 million instructions per second and cost about $10 million. For many image-processing operations, significantly improved cost/performance ratios can be realized using special-purpose architectures.

An important class of image-processing algorithms involve simple, local computations that are performed uniformly over an entire image array. For such algorithms, parallel-array processors such as the Goodyear Staran can be considerably more cost-effective than a general-purpose processor.

Staran is a single-instruction-stream, multiple-data-stream architecture, consisting of a conventional control unit for instruction sequencing and decoding up to 32 array modules [1]. Each array module consists of 256 simple processing elements that operate in parallel under one control stream. Each processing element has 256 bits of local storage on which it can perform arithmetic and search operations in conjunction with global operands. The Staran installation at ETL contains four such processor arrays and costs about $1 million.

Staran can perform local neighborhood operations at least one to two orders of magnitude faster than current generation serial processors. For example, Staran can convolve a 256 x 256 image with a 3 x 3 mask in 0.8 second, compared with 16 seconds on a dedicated CDC 3300 [2]. In this test, over 90 percent of Staran's execution time was consumed by I/O overheads associated with windowing in pieces of the image from disk. I/O overhead becomes a much less significant factor in computations involving a large number of operations per pixel. Staran can convolve a 1024 x 1024 image with a 13 x 13 mask in 133 seconds, a computation estimated to take several days on the CDC machine. It thus appears that Staran is optimally suited to medium-size neighborhood operations involving many operations per pixel.

Recently, a third type of processor has been introduced, specifically designed to perform local neighborhood operations efficiently on large image arrays. Known generically as video stream processors (VSPs), they are capable of executing many common image-processing functions faster than Staran, and at a fraction of the cost.

III VIDEO STREAM PROCESSORS

A VSP consists essentially of a large, fast bulk memory with sufficient capacity to store several entire images, a high-speed bus, and one or more high-speed serial processing units. Image data are continuously read out from the memory in a raster format and placed on the bus. The data are streamed through a selected processing unit at video rates (100 nanoseconds/pixel for a 512 x 512 image). The results are then output to a video display, stored back in memory, or pipelined to another processing unit.

Simple computations (e.g., primitive arithmetic and logical operations) can be performed in one pass though image memory, using a low-cost, general-purpose arithmetic logic unit (ALU). More complex computations, such as histogramming and convolution, can be decomposed into simple steps that are performed on successive passes. Where the expense is warranted, many such operations can be performed in a single pass, using special-purpose processing units and pipelining.

Although the inherent 10 MIPS serial-processing rate of a VSP is modest by supercomputer standards, total throughput is optimized by capitalizing on two design features: elimination of I/O bottlenecks by keeping an entire image in fast memory and elimination of software overheads associated with address indexing by delegating this function to the memory hardware. For local neighborhood operations on images, these two factors typically account for nearly 50 percent of total execution time on a conventional, general-purpose computer and (as we have seen) as much as 90 percent on Staran.

Historically, VSPs evolved from a marriage of two technologies: raster graphics and digital processing. The earliest raster graphics displays, such as the RAMTEK GX-100, were designed solely for viewing digital imagery. Second-generation displays, such as

the Comtal 8000 series, included high-speed function memories and lookup tables that could dynamically transform stored pixel values before display (Figure 1). This capability allowed image enhancements (such as contrast stretching and pseudo-coloring) to be performed in real time under interactive control. Third-generation display systems, such as the G.E. Image-100 and Stanford Technology 70/E [3], brought the first real processing capabilities. Using lookup tables, combinational logic, and output function memories, elementary arithmetic and logical operations could be performed at video rates on corresponding pixels in two or more images. The results could be directly displayed or fed back to an image channel for subsequent transfer to a host computer. (See Figure 2.) This pixel-processing capability was useful in a variety of remote-sensing applications, including change detection, spectral ratioing, and classification. For example, a land use category could be assigned t every pixel location by forming a weighted sum of the corresponding pixel values in several spectral bands and then passing the result through an output function table that mapped it into a use category.

The above pixel processors lacked several key features required for general image-processing operations. Most notable were:

(1) The ability to perform arbitrary arithmetic and logical functions.

(2) The ability to shift images relative to each other and thereby synthesize spatial neighborhood operations, such as convolution and edge detection.

(3) The ability to accumulate results globally over an image, as required for histograms, statistics, and plane fitting.

(4) The ability to make processing at each point in an image conditional on results computed at that point on a prior pass.

In 1976 SRI proposed a conceptual design for a processor, based on these extensions of existing raster display technology. SRI then publicized the design at several image-processing workshops* in hopes of

---

* Engineering Foundation Workshop on Image Processing, Rindge, New Hampshire, August 1976; ARPA Image Understanding Workshop, University of Southern California, October 1976.

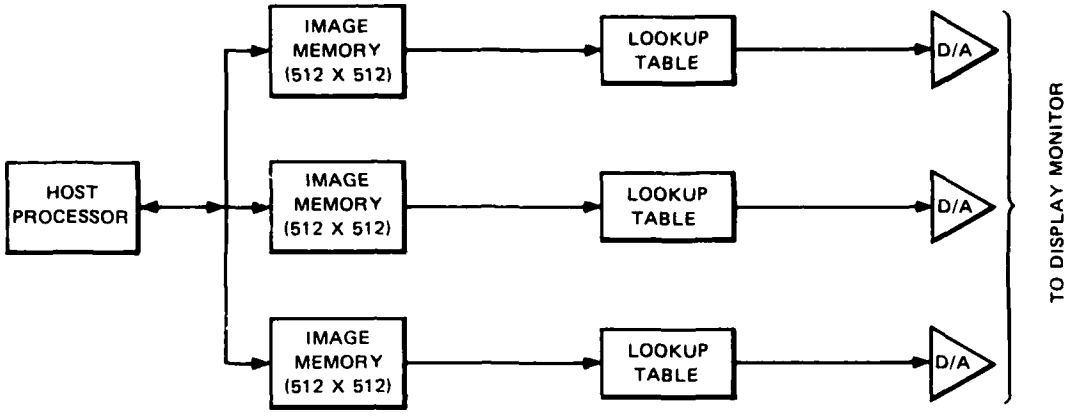


FIGURE 1 SIMPLIFIED DIAGRAM OF EARLY DISPLAY PROCESSORS

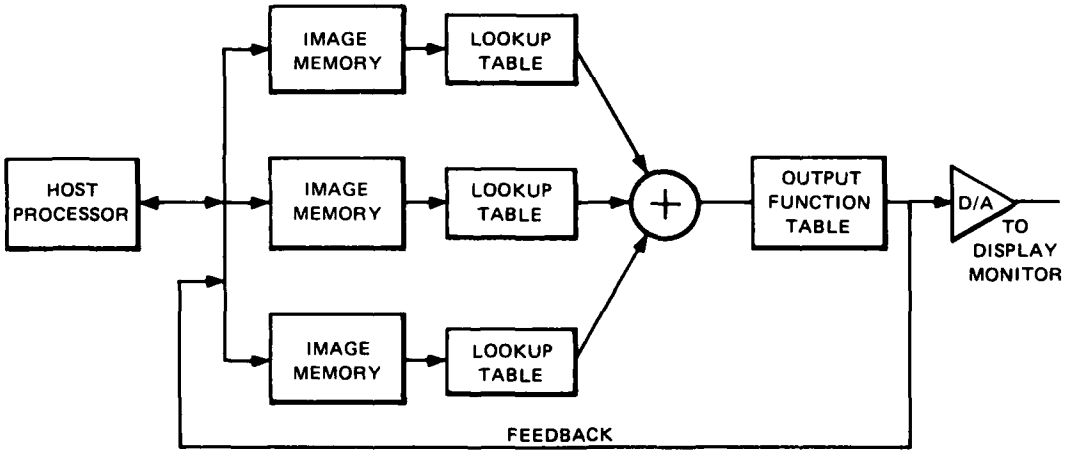


FIGURE 2 SIMPLIFIED FUNCTIONAL DIAGRAM OF A TYPICAL THIRD-GENERATION DISPLAY PROCESSOR

interesting industry attendees in building it. A local company, De Anza Systems, requested detailed specifications and 18 months later demonstrated the prototype of their IP-5000 image-array processor. Although the IP-5000 was a considerably scaled-down version of the proposed design, it is still, in many respects, the most advanced VSP commercially available.

In 1978 SRI acquired the first production IP-5000 for use in Advanced Research Projects Agency (ARPA)-sponsored research on image understanding. The Engineer Topographic Laboratories (ETL) agreed to fund an evaluation of the system's potential for cartographic application. We begin this evaluation with a description of the generic VSP design proposed to De Anza and others in 1976. Our purpose is twofold: to highlight the fundamental concepts and application potential of this class of machines, unconstrained by design limitations of a particular product; and to provide a standard against which the IP-5000 and other commercial products can be compared.

A. Generic VSP Design

Our proposed VSP design has a modular architecture consisting of memory and processors interconnected by a high-speed bus (see Figure 3).

Memory is organized as a large number of two-dimensional bit planes that may be addressed individually to store binary arrays or in stacks to store images. (Stacks can have arbitrary depth, providing efficient storage for images quantized to different precisions.) Each array is scanned in a continuous, top-bottom, left-right raster, with one pixel available at each point in time. The memory-addressing logic allows different offsets and sampling intervals to be specified for each array window, so that arrays, in effect, can be scrolled and zoomed with respect to each other. Although single-pixel access suffices for most operations, with optional delay lines a 3 x 3 neighborhood of pixels can be made accessible at each image location.

One port of the memory is connected via a fast bidirectional bus to a variety of processors, as well as image input and output devices

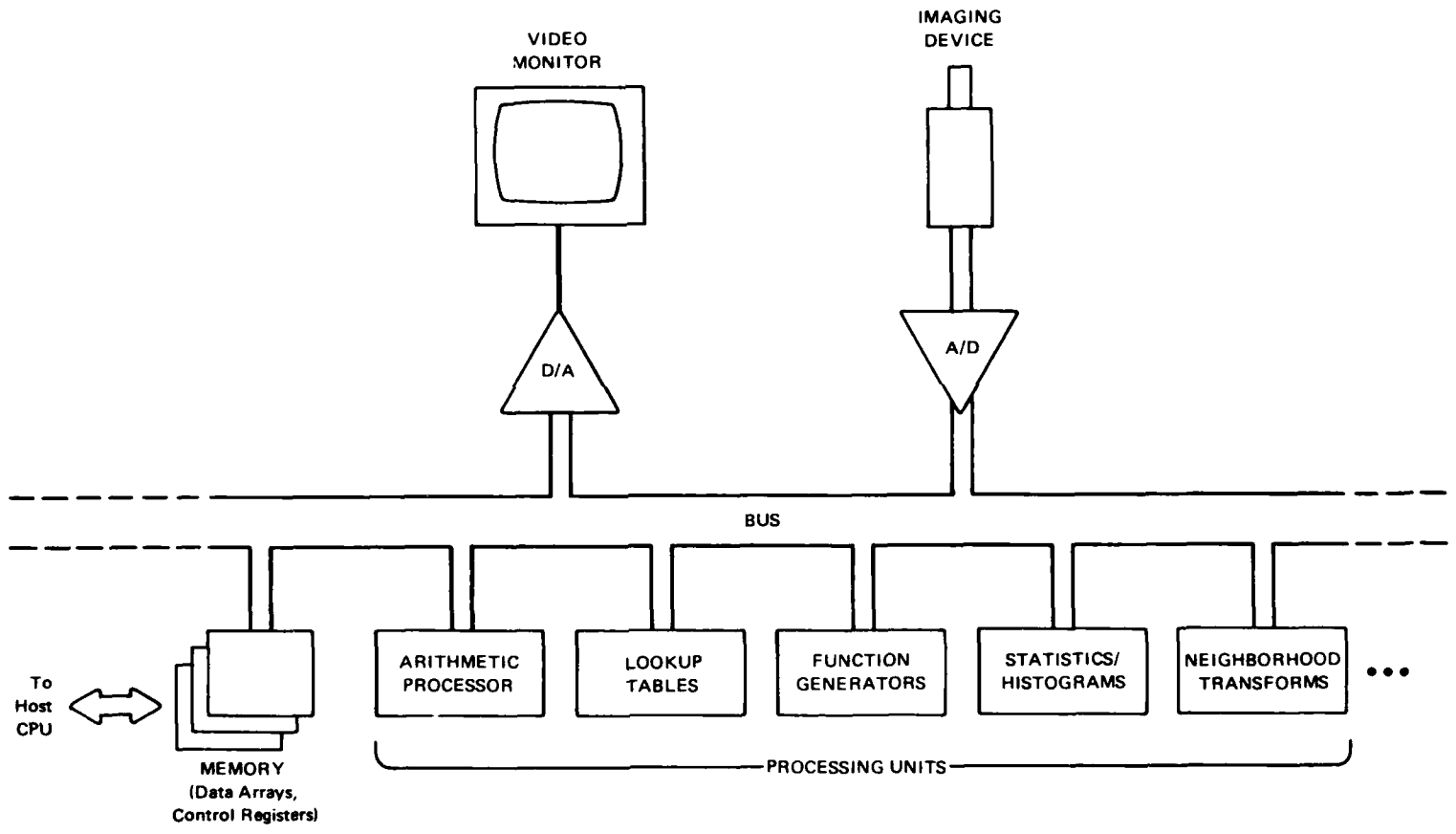


FIGURE 3 VSP ARCHITECTURE: MEMORY AND PROCESSING UNITS INTERCONNECTED BY A HIGH SPEED BUS

(raster display, image digitizer, etc.). The output of selected memory planes is routed into a processor whose output may then be written back into memory or input directly to another processor (pipelining). Memory and processors may thus be regarded as functionally equivalent.

A second memory port is connected directly to the memory bus of a host processor. The host loads images into VSP memory from a disk file and controls VSP operation through either an interactive user interface or direct subroutine calls from programs written in a high-level language. If the host has an adequate virtual address space, it can directly address VSP arrays as an extension of its physical memory. Thus, high-level sequential computations performed by the host can be efficiently intermixed with low-level parallel operations performed by the VSP.

A number of VSP control and data registers are accessible to the host. Control parameters that may be specified include the processor and sources of data to be used in a computation, the destination of the result, and the values of registers that delimit subareas of an image to be processed. Data registers that can be accessed upon completion of a computation include various accumulators and event counters that are used to compile statistics over an image or record extremal values. Some of the more important control parameters and data registers are listed in Table 1.

B. Processors

The VSP architecture just described can accommodate a wide variety of processing modules. At one extreme are inexpensive bit-sliced ALUs that can synthesize arbitrary arithmetic and logical functions over many passes through an image. At the other are expensive and highly specialized logic arrays designed to perform complex operations such as convolution in a single pass. Three modules were proposed as the basis of a general-purpose, image-processing facility: function generators, lookup tables, and a general-purpose processor. Used in conjunction with the spatial offset features of memory and the conditional

Table 1

CONTROL PARAMETERS ACCESSIBLE TO HOST PROCESSOR

****

Data Source(s):	Memory planes or processors to be used as data sources for a computation. (The number of sources depends on the processor.)
Data Destination(s):	Memory planes, processor, D/A converters (display). Multiple destinations can be designated.
Memory Address Parameters:	Zoom and offset can be independently set for each memory plane, but must be the same for both reading and writing.
Processor:	Selects processor to be used in computation.
Processor Parameters:	Initialize computations, select processing options.
Mask Planes:	A register specifying a set of memory planes that are ANDed together to form a binary mask delimiting the area of an image to be processed.
Window Register:	A set of registers specifying the coordinates of a rectangular window, used in conjunction with the mask planes to further constrain the area of processing.
	By restricting processing to specified subimages within a memory plane, the mask planes and window register allow conditional processing based on previous results.

****

processing provided by mask planes and window register, these three modules suffice to implement most low-level, image-processing algorithms. Special-purpose modules can then be added as needed to optimize the performance of particular operations.

1. Function Generators

Function generators are inexpensive processing units that can be used in lieu of memory as the source of simple spatial functions (e.g., constants, linear ramps, sinusoids) needed in many computations. Instead of storing such functions as images, they can be generated on the fly as functions of the current X,Y image coordinates. Binary masks (e.g., straight lines, circles, filled polygons) can be similarly generated, without wasting expensive storage.

2. Lookup Tables [Stack . Table => Stack]

Lookup tables provide a fast, inexpensive way to implement local unary image functions such as logarithmic scaling, multiplication by a constant, and thresholding. A table is first initialized by storing the desired output value corresponding to each input value. During execution, pixel values streaming through the processor are used as indices to select replacement values from the table. Lookup tables must have 512 addressable slots (9 bits) to be able to scale x,y coordinate values. These tables can be initialized by the host processor in the vertical retrace interval between frames.

3. General-Purpose Processor

An essential requirement is a processing unit that can perform arithmetic and logical operations on pixels from two source images (or function generators) as they stream through at video rates.

The architecture of the proposed processor is sketched in Figure 4, and its capabilities are outlined in Table 2. Arithmetic, logical, and test operations are performed at video rates on data from two image stacks (up to 32 bits deep) or from an image stack and an

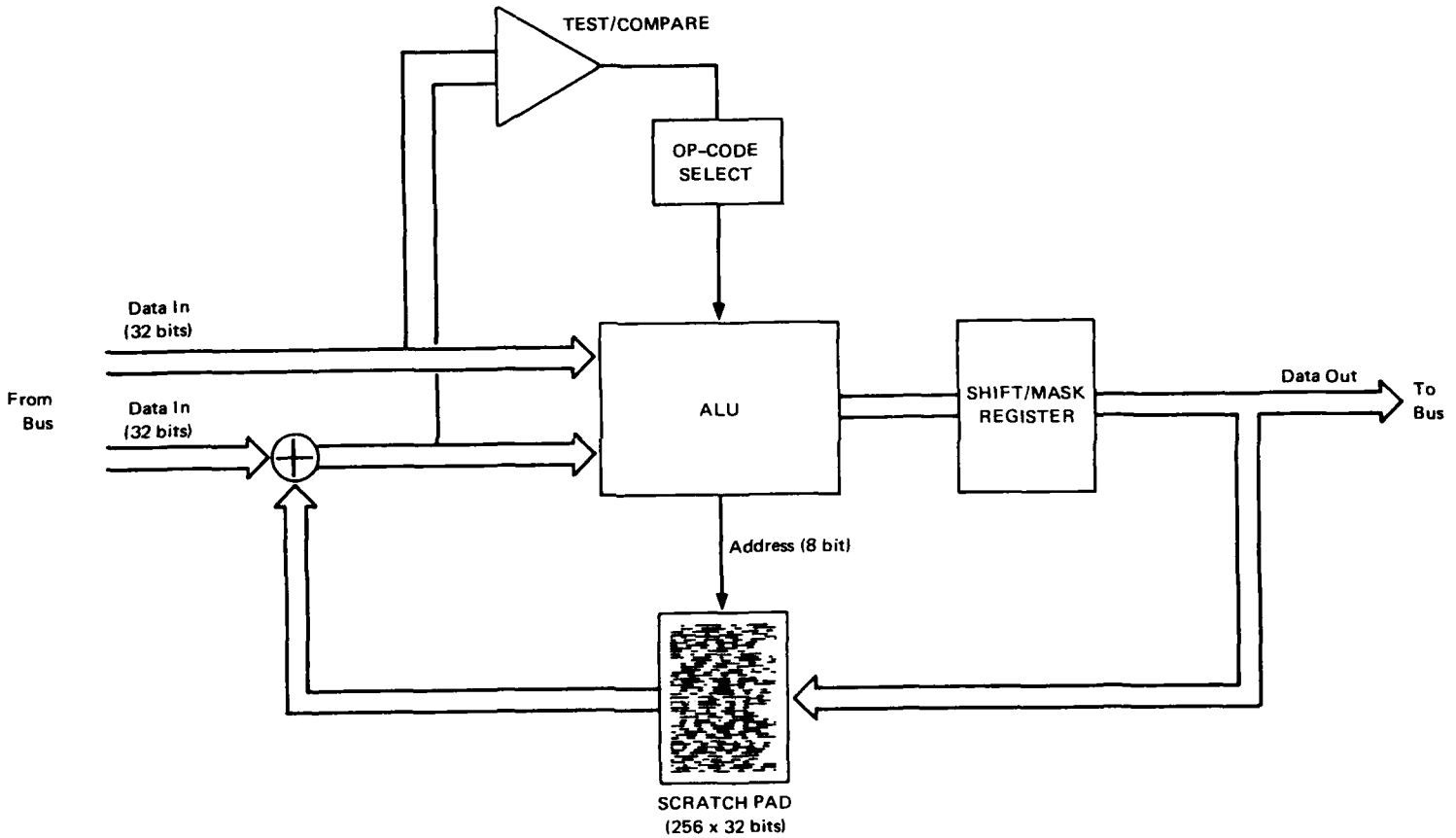


FIGURE 4 ARCHITECTURE OF GENERAL-PURPOSE PROCESSING MODULE

Table 2

SUMMARY OF PROCESS FUNCTIONS

Logic Operations [Bit Plane(s) A . Bit Plane(s) B => Bit Plane(s) C]

A, NOT A, B, NOT B, A AND B, A NAND B,
A OR B, A NOR B, A EXOR B

Logical Shift
SHIFT LEFT/RIGHT (n) filling with zeros or ones,
ROTATE LEFT/RIGHT (n) with wrap-around

Arithmetic Operations [Stack . Stack => Stack; Stack . Register => Stack]

A+B, A-B, A*B, A/B,
ASHIFT A(2^n), MIN (A,B), MAX(A,B), ABS(A) & overflow detection

(Real-time multiplication and division are limited to 9-bit numbers: 9 bits are required to handle x,y coordinate values; two 9-bit numbers can be multiplied and the result accumulated in a 32-bit sum in 1 frame time.)

Test [Stack => Bit Plane; Register => Bit Plane]
ZERO A, NZERO A, POSITIVE A, NEGATIVE A; same for B.

Compare [Stack . Stack => Bit Plane; Stack . Register => Bit Plane; Register . Register => Bit Plane]
EQUAL, NEQUAL, GREATER, LESS, >=, <= .

(Test and compare operations; allow branching by selecting which of two operations to perform at each pixel location.)

internal register. The result can then be logically shifted up to 32 bits, masked, and stored back in an image stack or a register. 256 32-bit registers are provided for compiling global sums required in histograms, moments, and similar computations. Standard 8-bit pixel values can be used to index the registers in operations such as histogram compilation.

IV USE OF VSPs

A VSP can be used in an image-processing system in two basic ways: as an extension of a host CPU's physical memory and as a peripheral processor for offloading low-level operations. The first use provides substantial speedups in conventional sequential algorithms because entire images can be stored in main memory, eliminating disk latencies. The second use provides additional speedups for pseudo-parallel operations and is our main interest here. Since many algorithms involve both serial and parallel aspects, the ability to intermix processing by the host and by VSP modules on the same data arrays is a very important feature.

A. General Programming Considerations

Before discussing specific algorithms, we introduce a few general considerations that arise in programming this class of machine.

A VSP can be viewed as a serial simulation of a single-instruction, multiple-data stream (SIMD) parallel-array processor, such as Staran. On each pass through an image (iteration), the same computation is performed at each pixel location, except where inhibited by a mask array. Each computation has access only to data in a local window centered on the currently addressed pixel (typically the eight immediately adjacent neighbors) and must be completed within the available time window. Temporary storage is available at each pixel location via auxiliary image stacks.

Alternatively, a VSP can be viewed as a conventional serial computer, operating on sequential streams of pixels. Programming is simplified, however, because addressing loops are eliminated. The input data registers will always contain the pixel values at the current coordinates of the source images; similarly, the contents of the output

registers will always be deposited at the appropriate coordinates of the destination images. Thus, a simple three-line loop of the form Load A, Add B, Store C suffices to add two images and store the result in a third image. The input and output registers, in effect, act as moving windows over images that are being streamed past them.

From either view, VSP programming involves computations on array operands. The computations must be partitioned into a sequence of steps, each of which can be performed on local windows within a 100-nsec interval. In the following programming examples, we shall assume that word and register sizes are sufficient to avoid problems of precision and overflow, and also that adequate image memory is available for storing temporary results. Such considerations can greatly complicate programming and cannot be taken for granted in practical implementations.

B. VSP Implementations of Common Image-Processing Algorithms

What class of image-processing algorithms are suitable for VSP implementation? Generally speaking, the answer is any function that can be implemented as a local iterative or pseudo-parallel computation. This class includes many of the most commonly used low-level operations for tasks, such as image smoothing, edge enhancement, feature extraction, and feature classification, as well as some computationally intensive parts of higher-level operations, such as region analysis and relaxation labeling. It excludes operations that require random memory access and/or complex local decisionmaking, such as edge tracking, geometric warping, and object recognition. Such operations can be implemented more efficiently on the host computer, using the VSP as bulk memory. Also excluded are unitary transforms (e.g., Fourier, Hadamard, Karhunen-Loeve, etc.), at least in frequency bands that require the convolution of two full-size images. Such transforms are best handled by special-purpose array processors that can share memory with the VSP.

Based on our own familiarity with the field, an extensive literature survey, and discussions with ETL personnel, we have compiled

a representative list of functions that meet the criteria for effective VSP implementation and are also of interest to the cartographic and image-understanding communities (see Table 3). The following sections outline implementations of these selected functions on the generic processor described in the previous section.

1. Primitive Image I/O Functions

a. Image Digitization

The high-speed A/D converter connected to the memory bus allows images from a television camera or other scanning device to be digitized at video rates and stored in an image array. Alternatively, the output of the A/D can be routed through the ALU processor and added to preceding frames in real time to reduce noise.

b. Image Copy

An image in one stack can be copied by routing it through the ALU and storing it in (one or more) other stacks.

c. Spatial Image Transforms

The ability to shift and magnify an image is essential for both processing and display. Images are shifted horizontally or vertically (with truncation or wraparound) by introducing appropriate offsets in the memory address counters. Zooming is accomplished by dividing address counters by powers of two and replicating pixels at intermediate screen locations. The resulting blocky appearance at high magnification can, if necessary, be improved with special-purpose hardware that performs bilinear or cubic-spline brightness interpolation in real time.

Shift and zoom can be altered dynamically on each iteration, interactively or under program control. This capability can be used to increase the effective number of image planes at the cost of resolution. A 512 x 512 array can be used, for example, to store four

Table 3

SUITABLE VSP OPERATIONS

Primitive Image I/O Functions:
- Image digitization
- Image copy, scroll, zoom
- Graphics overlays
- Gray-scale mapping (pseudo color)

Primitive Pixel Operations:
- Logical (AND/OR)
- Arithmetic
  - + - * /
- Conditional
  - compare two images
  - test image (greater, less, equal to a constant)

Arithmetic, logic, and conditional operations, together with image copy and scroll, are sufficient to synthesize any image-processing function, given sufficient time and memory.

Statistics:
- Histogramming
- Max, min, sum, avg, count of masked points
- Moments (sum of 1, Z, X, Y, X2, Y2, XY, etc.) over image for masked points
- Line and plane fitting
- Statistics of texture operations (directed edge density)
- Hough Transform (broken line detection)

Classification:
- Ratioing of spectral bands (to normalize illumination)
- Linear combination of images
- Nearest neighbor classification of linear combination

(These three steps constitute classical LANDSAT processing.)

Local Neighborhood Operations:
- Binary pattern transformations
  - noise cleaning, gap filling, thinning, thickening
- Arithmetic transforms
  - averaging, median filtering, simple edge detection, and texture operators (e.g., Sobel)

Convolution, Correlation:
- Weighted averaging, spatial filtering, template matching
- Relaxation via correlation
- Complex edge detection (e.g., Hueckel)

Propagation Algorithms:
- Distance transforms (chamfering)
- Skeletonization, shape abstraction
- Connected component labeling (regions, edges)
- Association algorithms (sketch completion)

Image Generation:
- Binary mask generation (e.g., lines, circles, boxes)
- Shaded surface generation
- Hidden surface suppression
- Image masking, merging, mosaicking (insertion)

Unitary Transforms:
- Fourier, Hadamard, Karhunen-Loeve
  - (performed by peripheral-array processor that shares memory with VSP)
- Transform domain filtering
  - (convolution by multiplication of Fourier-transformed images)

Real-Time Imaging Processing:

Image Enhancement
- Photometric compensation
  - scale, offset, trend
  - shading correction, time averaging (to reduce noise)
  - gray-scale statistics
  - histogram modification (contrast stretching, compression)
- Accommodation
  - control of focus, iris

Motion Analysis
- Motion, change detection (difference and threshold)
- Velocity map
- Centroid tracker

Hybrid Electro-Optical Processing
- Unsharp masking (defocus and subtract)
- Moving edge detection (subtract sequential images)
- Correlation peak detector
- Normalized Fourier signatures (a la Landeris and Stanley)
- Feature extraction for patterned lighting
  - (e.g., light stripe range finder)

VSP Applications in IU Systems:

- Scene partitioning (e.g., Ohlander region finder)
- Interactive overlay generation
- Road tracing
  - applying Duda road operator
  - chamfer (mask generation)
  - cost propagation (min cost determination)
- Map matching
  - feature extraction (convolution, edge detection)
  - computation of distance arrays (chamfering)
  - compution of average match error from chamfer array
  - scores for flexible template matching
- Intrinsic image reconstruction
  - local smoothing, differentiation, reintegration

256 x 256 images; each subimage would then be centered and zoomed to full size, prior to use. Simple real-time animations can be performed without excessive storage requirements by storing multiple views of an object in different subimages and accessing a different subimage on each iteration.

d. Gray-Scale Mapping

Gray-scale mapping and pseudo-coloring are performed by passing the output of an image memory through an appropriately initialized data transformation table. The transformed values can be stored in an image memory or used directly for display or further processing.

e. Overlays

The data transformation tables provide great flexibility in generating graphical overlays for superposition on displayed images. Any bit position of an image can be mapped into an arbitrarily colored, high-brightness overlay, using the transformation tables. Thus, an 8-bit image can specify up to eight independent overlay planes. Alternatively, a pixel can be interpreted as specifying one of 256 categories, each of which can be independently mapped into an overlay color.

2. Primitive Pixel Operations

Primitive arithmetic and logical operations on individual pixels or corresponding pixels from two images are performed using the ALU and lookup tables. Addition, subtraction, and simple logical functions (e.g., AND, OR) are performed in the ALU in one frame time. Multiplication and division can also be performed in a frame time if the ALU contains a high-speed multiplier chip; in the absence of such a chip, 8-bit multiplication can be synthesized in multiple passes with shifts and adds. Regardless, multiplication by a constant can be done in one pass using a lookup table, and multiplication/division by a power of two can be performed using the ALU logical shifter.

All of the above computations can be restricted to regions of an image designated by a mask array. Furthermore, the ALU can be programmed to execute one of two alternative operations at each pixel, based on the results of a test or comparison operation performed in the same iteration. Examples of such conditional operations appear below.

3. Statistics

Statistics, moments, and histograms play a vital role in many image-processing tasks. Statistics and histograms are used to determine intensity transformations that optimize display, as well as for texture classification. Moments are used to characterize the shapes of binary blobs, to fit lines and curves to binary data points, and to fit surfaces to gray-level image data.

Simple statistics such as the max, min, sum, avg, or count of a set of masked points can be computed by the ALU in a single frame time using registers to compile sums or extreme values. Simple moments of the point set (e.g., sigma x, sigma xy, and sigma x_2), as well as intensity-weighted moments (e.g., sigma Z, sigma Zx), can be similarly computed using multiply and adds, where one or both operands are pixel coordinates (provided by a function generator). Histograms can be compiled for a set of masked image points by using pixel values as indices to select which ALU register to increment.

More sophisticated statistics can be compiled on multiple passes. Statistics of local edge operators (see below), such as histograms of edge strength and direction, are useful as texture statistics. Histograms of projections of edge points onto lines are useful for detection of long (possibly broken), straight lines. One well-known technique, the Hough Transform, can be implemented as follows:

> First, an array of edge strengths is obtained by convolving an image with an edge operator. Second, each edge point (x, y) is projected onto a line through the origin with

slope t, by computing the function,

$$p = x * \cos t + y * \sin t \quad .$$

(Multiplication by constants cos t and sin t can be performed in lookup tables followed by summation in the ALU.)

P is used as an index to select an ALU register, which is then incremented by the edge strength at (x, y). A strong edge perpendicular to line t will thus cause a high value to be accumulated in a particular register.

The above projection process is then repeated on subsequent iterations for different slopes. (In time-critical applications, a special-purpose processor with a two-dimensional array of accumulators can be used to compute Hough Transforms in a single pass.)

4. Image Enhancement/Classification

VSP architecture is ideal for performing real-time, continuous image enhancement. Raw video input can be routed through the ALU, where shading correction can be performed (by subtracting an image of a blank field) and noise reduced (by time averaging with previous frames). Video can also be routed through lookup tables, where arbitrary scale and offset corrections can be applied, including those necessary to effect histogram equalization.

Conventional multispectral classification involves similar processing. The ALU can be used to compute images that are ratios of spectral bands and to form images that are linear combinations of such ratios. Lookup tables can then classify the resulting sums into previously established categories, using any number of decision rules, such as nearest neighbor or piecewise-linear discriminant.

5. Local Transforms

The previous sections all dealt with operations involving a single pixel in each operand image. Another large class of operations involve local neighborhoods of pixels (e.g., the 3 x 3 area surrounding each image location).

Binary pattern transforms operate on a Boolean array and replace each bit by a new value based on the surrounding pattern of bits in a 3 x 3 neighborhood. Such pattern transformations have long been used on serial computers to implement iterative binary image-processing operations, such as line thinning and thickening, gap filling, and isolated point noise suppression [4].

An efficient way to implement binary transforms on a VSP is first to convert the binary array into a 9-bit image stack, each pixel encoding the nine surrounding bits at the corresponding location in the original binary array. The encoded image is then passed through a lookup table that transforms it back into a binary array, replacing each surround with an appropriate Boolean value for the central bit at that image location.

Transforming 3 x 3 binary spatial patterns into an equivalent gray-valued image can be accomplished in eight passes through the ALU; on each pass, the binary array is spatially shifted (+1 or -1 in x and/or y) to pick up a bit value, which is then stored in the appropriate bit plane of the output array stack. The original binary image serves as the ninth plane. Passing the resulting image through the lookup table takes another pass. Thus, binary neighborhood transforms can be performed in nine frame times (270 msec for a 512 x 512 image) and iterative transforms in nine frame times/iteration.

Arithmetic neighborhood transforms operate on an image stack and replace each pixel with the result of a computation performed on the surrounding 3 x 3 neighborhood. These transforms are commonly used for spatial noise reduction (averaging), edge and texture feature detection, and filtering (e.g., unsharp masking).

Arithmetic transforms are implemented by passing the original image and spatially offset versions of it through the ALU. A simple transform such as replacing each pixel by the average of its neighbors takes nine passes. First, an 11-bit output image is initialized to zero. On each of the next eight passes, the input image is shifted one unit vertically and/or horizontally and added to the output image. On

the ninth pass, the output image is divided by eight (e.g., by right shifting each pixel three bits). Other transforms are similarly implemented. For example, replacing each pixel by the max or min of surrounding pixels can be accomplished by using the ALU first to copy the input image into an output image, and then to compare the copy with shifted versions of the original. The max or min resulting from each comparison is stored back in the output array. Thus, after eight passes, each pixel in the output image will contain the max or min value of each corresponding 3 x 3 neighborhood in the original image.

As a final example, we shall sketch a VSP implementation of a Sobel edge operator. Sobel's operator is defined on a 3 x 3 window,

$$\begin{matrix} a & b & c \\ d & e & f \\ g & h & i \end{matrix}$$

as

$$S = H + V \qquad \text{where,}$$

$$H = \text{ABS}\ ((a + 2b + c)-(g + 2h + i))$$
$$V = \text{ABS}\ ((c + 2f + i)-(a + 2d + g))$$

Ignoring minor improvements in efficiency, an obvious way of computing this function is to compute H and V independently at each point and then add the two images holding these partial results. To compute H, a lookup table is first initialized to multiply an image by the constant 2. The term 2f is computed by spatially shifting the input image left, passing it through the table, and storing the result in an output image serving as an accumulator. Terms c and i are added to 2f on the next two passes, by appropriately shifting the input image with respect to the accumulator image. On the next three passes, terms a, 2d, and g are obtained and subtracted from the sum, again making appropriate use of spatial offsets and the lookup table. To compute absolute values, the accumulated sum is again passed through the ALU, which tests the sign bit of each pixel and complements it if negative. Computation of V is performed analogously to H, and the final summing of H and V is straightforward.

Local transforms are good candidates for high-speed implementation with special-purpose hardware. Computational structures such as adder and logic trees can be used in conjunction with 3 x 3 memory access windows to perform many local transforms in a single pass.

6. Convolution and Correlation

Convolution and correlation are generalizations of local arithmetic transforms to larger neighborhoods. In essence, the value at each point in an image is replaced by a weighted sum of surrounding pixel values, with the weights specified by a local operator, subimage, or mask. These operations are widely used in image processing for spatial filtering, template matching, sophisticated edge detection (e.g., the Hueckel operator), stereo correspondence, relaxation updating, and many other functions. They are also computationally very expensive; convolving a 1,000 x 1,000 image with a 32 x 32 mask involves nearly a billion multiplies (about three hours on a modern general-purpose computer).

Like the Sobel operator, convolution can be implemented iteratively by passing appropriately shifted versions of the input image through a lookup table and then summing the resulting weighted pixel values in an output "accumulator" image. One iteration/mask point is required, at one frame time/iteration. Thus, assuming a 30-millisecond frame rate, an entire image can be convolved with a 32 x 32 mask in about 30 seconds, nearly 400 times as fast as on a general-purpose machine.

Relaxation updating of edge strengths can be implemented by iteratively convolving an edge array with an array of correlation coefficients and passing the resulting array of weighted sums through a nonlinear function implemented in a lookup table. After each iteration, convergence is tested in one pass through the ALU, by comparing the array of updated values, pixel by pixel, with the results of the previous iteration; the maximum difference is retained in an accumulator and used as a termination condition.

7. Propagation Algorithms

Propagation algorithms are a class of local transforms that stores results back into the original input image for use by subsequent computations in the same iteration. Information thus propagates top-bottom and left-right in the direction of the raster. Distance transforms, skeletonization, and connectivity analysis are examples of algorithms commonly implemented by propagation techniques.

A distance transform starts with an array of distinguished feature points and produces an output array whose numbers represent distances to the nearest feature point. Distance arrays are important for shape matching and map matching [5], for shape abstraction (skeletonization), for sketching in broken boundaries [6], for designating points within a desired vicinity of some initial set, and for many other functions.

Propagation algorithms, by definition, require access to a neighborhood of pixels at each image location. Assuming access to a 3 x 3 neighborhood, a distance array corresponding to a given feature array can be generated in two passes [7 and 8]. The input feature array, (F[i,j] i,j=1...n), is initially two-valued: 0 for feature points and 2n otherwise. A forward pass (top-bottom, left-right) modifies the feature array as follows:

```
FOR i _ 2 STEP 1 UNTIL N DO
  FOR j _ 2 STEP 1 UNTIL N DO
    F[i,j] _ MINIMUM(F[i,j], (F[i-1,j]+2),
                (F[i-1,j-1]+3), (F[i,j-1]+2),
                (F[i+1,j-1]+3));
```

A backward pass (bottom-top, right-left) then completes the operation:

```
FOR i _ (N-1) STEP -1 UNTIL 1 DO
  FOR j _ (N-1) STEP -1 UNTIL 1 DO
    F[i,j] _ MINIMUM(F[i,j], (F[i+1,j]+2),
                (F[i+1,j+1]+3), (F[i,j+1]+2),
                (F[i-1,j+1]+3));
```

The incremental distance values of 2 and 3 provide relative distances that approximate the Euclidean distances 1 and the square root of 2.

To generate the skeleton of a shape, one would first generate a distance transform from its boundary points and then pass the resulting distance array through a local arithmetic transform that flags points that are local maxima.

In connectivity analysis, the problem is to identify sets of adjacent pixels with the same value and assign each set a unique label. This operation can be used, for example, in remote sensing to form spatially connected regions following pixel classification. It is frequently used in binary image processing to analyze spatial characteristics of blobs (e.g., size, shape), as well as to connect adjacent edge points into contiguous lines.

Connectivity labeling involves two arrays--the input array and a label array--and can be performed in two passes through the ALU. In the first pass, the value at each point, b, is compared with the values of previously labeled neighbors, c and a, as shown below:

```
  a
c b
```

If the value at b is the same as the value at a, then b is assigned the same label as a. Otherwise, if b has the same value as c, it is assigned c's label. If b has a different value than both its neighbors, a label count is incremented and b assigned the next number.

The second pass deals with a complication that arises when b has the same value as both a and c, but these pixels had previously been assigned distinct labels. This can happen, for example, in a U-shaped figure, where the top-down scan causes each arm initially to receive distinct labels. Such inconsistencies are detected on the second pass by testing each location to determine whether [value (b) = value(c)] and [label(c) =/ label (a)]. Where this condition holds, label c is equated to label b in a lookup table. The label image is subsequently accessed through this table, which dynamically transforms all equivalent labels into a common number.

After performing connectivity analysis, previously described techniques can be applied to the label array to determine the largest or smallest region and statistics on region size. Using the label array as a mask, moments and other shape features can be computed for any designated region (e.g., the largest one) for recognition purposes.

Many generalizations of distance transforms and connectivity-labeling algorithms can be implemented on a VSP. Quam has described a distance transform that computes an excellent approximation to true Euclidean distance, by propagating the coordinates of the closest feature point [9]. The algorithm also permits more reliable skeletonization, using as a criterion points at which the direction to the nearest boundary point is discontinuous. Fischler has described a generalization of connectivity analysis that assigns a common label to all similar points that are not necessarily contiguous, but are within a specified neighborhood [10]. He has also developed symbolic distance transforms that propagate labels, rather than distances, and are useful for functions such as sketching in a broken boundary and producing skeletons that are resistant to local boundary perturbations [6].

Propagation algorithms impose strict constraints on VSP design. Unlike local transforms, which are pseudo-parallel, propagation requires that the value of point i be available before processing point i+1. A consequence is that complex operations can no longer be synthesized on multiple passes. (Processing of point 2 would have to wait k passes until a value of point 1 was determined, processing of point 3 would have to wait another k passes for the value of point 2, and so on, thus defeating the pseudo-parallel nature of a VSP.) In computing a distance transform, for example, the ALU must be fast enough to find the minimum of 5 points in an L-shaped window and store back the result, before the scan progresses to the next pixel location. A special-purpose comparator tree must be used if the general-purpose ALU cannot meet this requirement.

A second requirement is the ability to scan the image array in both forward (top-down, left-right) and reverse (bottom-top, right-left)

rasters. Some memory architectures can be scanned in a reverse raster simply by running the memory address counters backwards. Where a reverse scan is not possible (e.g., because the memory interleaving used to obtain video bandwidths precludes it), then the image resulting from the forward pass must be serially inverted in the host computer. The inverted image is then copied back into the VSP, which implements the backward pass.

8. Image Generation

Synthetically generated images are widely used in image processing as templates and masks, as well as for graphical presentation of results. A VSP can perform many of the image generation functions of expensive, special-purpose display processors, at comparable speeds.

A variety of binary masks can be readily generated. Straight lines, for example, can be drawn in one frame time, by passing the x and y coordinate values of each image location through the ALU and setting bits in a binary output array wherever a desired linear relation Ax+By+C holds. Areas within a specified distance of the line can be masked off by applying a distance transform to the binary array and then passing the resulting distance image through a lookup table that thresholds at the appropriate distance. Circles can be drawn at specified locations and radius, by initializing center points in a binary array and running a distance transform. The resulting distance array is passed through a lookup table that can be initialized to mark bits at locations where distance equals the desired radius. (A filled-in circle can be generated by initializing the table to mark locations where distance is less than or equal to the desired radius.)

Gray-level images can be masked, merged, and mosaicked to obtain useful image products. Image masking can be accomplished by selectively copying an image into a blank array, under control of a mask array. (Writing is inhibited where the mask is zero.) Similarly, images can be mosaicked by selectively copying one image into another under control of a mask. Two images can be merged, for example, by

passing both through the ALU and outputting the larger pixel value at each point. Despite their simplicity, such image manipulations are difficult or impossible to perform on conventional image-display processors, due to the lack of test and compare operations.

Shaded surfaces can be synthesized from an array of surface normals (e.g., a terrain map), using Horn's concept of a reflectance map implemented in a lookup table [11]. The lookup table maps each orientation into an appropriate brightness, based on the angle of incidence for an assumed light source. Given surface descriptions in the form of range arrays, a three-dimensional scene can be synthesized as follows: first, the range array for surface one is differentiated (e.g., by convolving it with an edge mask) to obtain an orientation array; the orientation array is transformed through a reflectance map to compute a brightness array; orientations and brightnesses are then determined for surface 2; and a test is made to determine locations where range (2) < range (1). Where this condition holds, brightness (2) replaces brightness (1) in the brightness array, and range (2) replaces range (1) in the range array. In this fashion, range and brightness arrays can be synthesized for a complex scene with multiple occluding surfaces, in a time linearly proportional to the number of surfaces.

9. Real-Time Image Processing

VSPs, operating at video frame rates, can perform a variety of processing on real-time continuous imagery. In such applications, a current frame from the A/D converter is processed in conjunction with previous frames stored in memory.

An important class of real-time processing applications concerns image enhancement. Two such techniques, real-time frame averaging for noise reduction and real-time photometric compensation (scale, offset, and camera shading), have already been described. Real-time camera control provides other opportunities [12]. Brightness statistics can be computed on incoming imagery and used to control camera iris, electronic sensitivity, and digitizer range. Edges can be

rapidly extracted (e.g., by subtracting an image from a spatially shifted version of itself), summed, and used as a sensitive criterion for optimizing camera focus.

A second important class of real-time processing involves analyzing imagery of moving scenes. Simple motion detection can be implemented by subtracting consecutive frames in the ALU and thresholding the result. Monitoring for changes or anomalies is similarly accomplished by comparing the current image with a stored prototype. Real-time centroid tracking can be performed by thresholding the incoming image in a lookup table and then computing the center of mass of the resulting blob as described in the section on statistics. For sufficiently slow movement, correlation tracking of distinguished features (e.g., corners) can also be implemented.

A velocity map of the image can be determined from local spatial and temporal gradients (computed by comparing shifted and consecutive images, respectively). Major moving regions in the image can then be identified from global statistics on this map accumulated in the ALU [13].

A third important class of real-time applications involves hybrid electro-optical processing. Optical-processing techniques provide a cost-effective solution to many image-processing problems, particularly those involving spatial filtering. Digital processing provides complementary capabilities in performing nonlinear operations and in interpreting results. The real-time capabilities of a VSP are well matched to the instantaneous response of optical systems.

A simple example of combined optical and VSP processing is unsharp masking, which can be implemented by subtracting a defocused (i.e., optically low-passed) image from a focused one. A second example involves optical-matched filtering; an image can be correlated against an array containing multiple templates, such as letters of the alphabet. The result is a correlation array with brightness peaks at locations corresponding to masks with good matches in the image [14]. This array can be processed in real time by a VSP to determine the locations of

correlation peaks, and thus the identities of masks with high correlation scores. See Gara [15] for related applications in industrial automation.

Fourier image filtering, as proposed by Lendaris and Stanley [16], provides a somewhat more sophisticated example of using a VSP to interpret the results of optical processing. Here, a Fourier transform of an image is obtained using coherent optics and the resulting array input through a video camera to the VSP. Scale and rotation invariant signatures of the image can then be obtained by summing the energy, respectively, in concentric rings about the center of the image and in pie-shaped wedges emanating from the center. These sums are easily obtained in a single pass; an image is first initialized with numbers that indicate which accumulator to increment when energy is detected at the corresponding location in the Fourier array. This mask array and the Fourier array are then passed through the ALU, and the signature accumulated in a fashion analogous to the computation of histograms.

Agin [17] has described the use of a triangulation range finder using a camera and a stripe of light. A VSP can detect the light stripe in the image (using thresholding) and do a least-squares fit (as described under statistical computations), all in a single pass. The resulting fit can be used to determine planar surfaces and for computation of range values.

C. Image-Understanding Techniques

We hope the foregoing sections have conveyed the wealth of possibilities for exploiting VSPs in low-level image processing. VSPs also have a role in expediting computationally intensive parts of higher-level, "image-understanding" (IU) algorithms. We have already suggested how a VSP could be used in such IU operations as relaxation labeling and skeletonization. In this section, we present three representative examples of how a VSP could be used in conjunction with a conventional serial computer to implement IU techniques.

1. Segmentation

Partitioning an image into homogeneous regions is an important step in many image-matching and object-recognition methods. One of the most effective approaches for partitioning complex images is the recursive decomposition algorithm developed at Carnegie-Mellon by Ron Ohlander [18]. Ohlander's input was three image arrays representing scene luminescence through red, green, and blue filters. His output was an array of region numbers suitable for input to a connectivity analysis routine. Five major processing steps were involved.

(1) Compute standard color coordinate transformations (X,Y,Z; hue, luminescence, saturation, etc.) by linearly combining the input images.

(2) Compute histograms of the original and transformed arrays.

(3) Determine the histogram with the most bimodal character.

(4) Partition the image into two exclusive classes corresponding to these dominant modes.

(5) Recursively partition the image by repeating Steps 2-5 for each partition, until the histogram for any partition is unimodal.

It should be obvious that all steps except Step 3 are directly implementable on a VSP. Recursive partitioning, in particular, is accomplished using a lookup table to assign new region numbers consistent with bimodal partitioning, and then using the resulting region label array as a mask to limit the set of pixels considered in subsequent histogram compilations. Up to 512 distinct regions can be handled, limited by the size of the lookup tables.

On large images, a conventional implementation of Ohlander's algorithm can be very time-consuming. The original version of the program running on a PDP-10 took nearly a week of overnight batch processing to segment a single 600 x 800 image. Thus, the speedups obtainable with a VSP are very significant.

2. Interactive Generation of Symbolic Overlays

A cartographic problem closely related to general segmentation is that of producing symbolic overlays that denote areas of an image containing some feature of interest (e.g., water, vegetation, sand, etc.). It has been observed [19] that while completely automated overlay generation is beyond the state of the art, an interactive refinement process can significantly reduce the amount of labor involved. The basic approach, as explicated in [19], involves using graphical interaction (e.g., light pen) to indicate a few examples and counter-examples of the desired feature in an image. Feature detectors (brightness, color, texture operators) are then applied in these regions and a discrimination criterion devised (automatically or manually). The operators and discrimination criterion can then be applied to the whole image (and neighboring images of similar terrain) to produce detailed overlays quickly.

Once again, all the basic steps in this approach, except the formulation of a decision criterion, are candidates for VSP implementation: generating binary masks corresponding to manually encircled regions, applying operators to the masked regions, and classifying pixels based on arithmetic and logical combinations of operator responses. Similar approaches to overlay production have been implemented previously using analog video technology and pixel-oriented display processors. In both cases, generality was limited to features that could be discriminated by thresholding linear combinations of point attributes (e.g., brightness in different spectral bands). With VSPs, classification can be based on spatial features such as texture density and blob shape, affording significant extensions in the range of features that can be discriminated. While such operators can be implemented on conventional computers, the speed of a VSP is essential for the interactive nature of this application.

An example of interactive overlay generation using the De Anza processor is presented in a following section.

3. Interactive Road Tracing

The interactive road-tracing algorithm developed under this contract is an excellent candidate for VSP implementation [20]. The major steps are:

(1) Interactive guideline specification.

(2) Generation of a search mask surrounding the guideline.

(3) Application of road operators within the masked area.

(4) Iterative computation of path scores based on local road strength.

(5) Tracing the optimal path between designated start and end points.

The first three steps make direct use of capabilities discussed in earlier sections. The computation of path scores starts with an array containing high cost values everywhere except at the starting point of the road, where cost is set to zero. The cost at each point is then iteratively updated to be the minimum of its present value or the cost of any surrounding point plus the incremental cost associated with the road score to that point. This iterative updating can be implemented either as a relaxation-updating algorithm or a propagation algorithm. Even optimal path tracing, which is nominally a sequential process, can be facilitated with a local operator that marks all points that are less than all but one of their neighbors, a condition necessary for points lying on the optimal path.

Many other IU algorithms could have been selected as examples; the parametric correspondence [5] and intrinsic image recovery [21] techniques developed at SRI are two obvious possibilities. It has been our experience that, with a little thought, most of the important algorithms developed throughout the IU community could be adapted to take advantage of VSP capabilities.

V OVERVIEW OF DE ANZA IP-5000

VSPs with the capabilities of our generic design, while technologically feasible, have yet to be produced commercially. De Anza System's IP-5000 perhaps comes closest. This product was designed in 1976, partially in response to specifications from SRI, similar to those in Section III A, and partially in response to requirements for the TEXAC texture analyzer proposed by Robert Ledley of the National Biomedical Research Foundation. In this section we present an overview of the IP-5000 and then critique it against the generic VSP described in Section III.

Figure 5 is a simplified data flow diagram of the IP-5000. The solid-state refresh memory is partitioned into four channels, each containing an 8-bit image at 512 x 512 resolution. Pairs of channels can also be processed together as a single 16-bit image for operations requiring that precision. The random-access MOS memories are dual ported and can be accessed either by the video processor or a PDP-11 host computer (through memory-mapping hardware). De Anza memory can thus be used by the PDP-11 as fast bulk storage for sequential computations.

Image data initially enters a memory channel from either the host computer or the video digitizer (via the processor). The data stored in each channel are continuously streamed through dedicated lookup tables and thence to D/A converters for display. The transformed data are also routed back to the video processor. A multiplexer at the input of the processing unit selects which memory channels will serve as operands for the current computation. A second switch at the output selects which channels will receive the results. (The contents of other channels remain unchanged.) Each image channel can be spatially shifted vertically and horizontally with respect to other channels so that

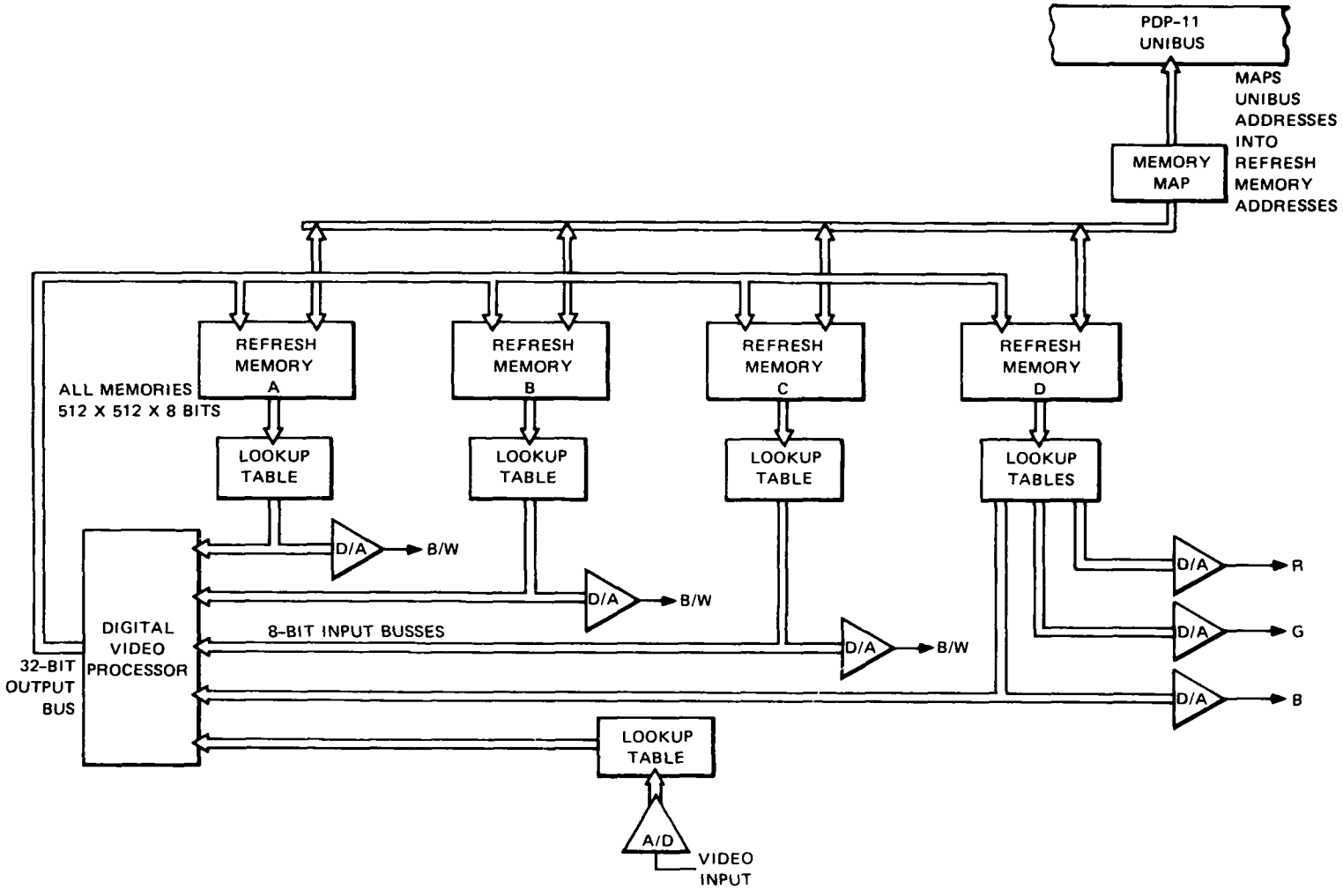


FIGURE 5 SIMPLIFIED TOP-LEVEL DATA FLOW OF DEANZA IP-5000 DISPLAY SYSTEM

complex neighborhood operations can be synthesized, using the processor. Each channel can also be independently magnified by powers of two, using pixel replication.

Figure 6 is a simplified functional diagram of the video processing unit. The architecture permits any two of a possible five arrays to be selected as inputs to each of the two arithmetic/logic units (ALUs). A fifth input is provided for data from function generators, although only a constant register is available on SRI's system.

The ALUs can compute most unary or binary arithmetic or logical functions of their input in real time. They can operate independently or in a pipelined mode, where the output of the lower ALU at a given image location determines which of two Op Codes will be executed by the upper ALU at the corresponding point in its input stream. Such pipelining allows conditional executions and is also used for carry propagation when performing 16-bit arithmetic.

The outputs of both ALUs can be routed to any of the four memory channels. For flexibility, the ALU outputs are combined in a shifter array that permits up/down shifts, end around or tested for overflow, either as a 16-bit word or two 8-bit bytes.

A 32-bit mask register permits the shift register output to be written selectively into memory planes of the designated output channels. Moreover, results are written only for those pixel locations contained within a specified rectangular window and enabled in a mask plane. (The mask is obtained from the high-order bit of the lookup table connected to the fourth image channel and can thus correspond to any of up to 256 distinct subregions.)

For statistics gathering and related applications, the upper ALU result for each enabled pixel is accumulated over the image; an event counter, connected to the carry-out of the lower ALU, can be used to keep track of the number of pixels contributing to the accumulated total.

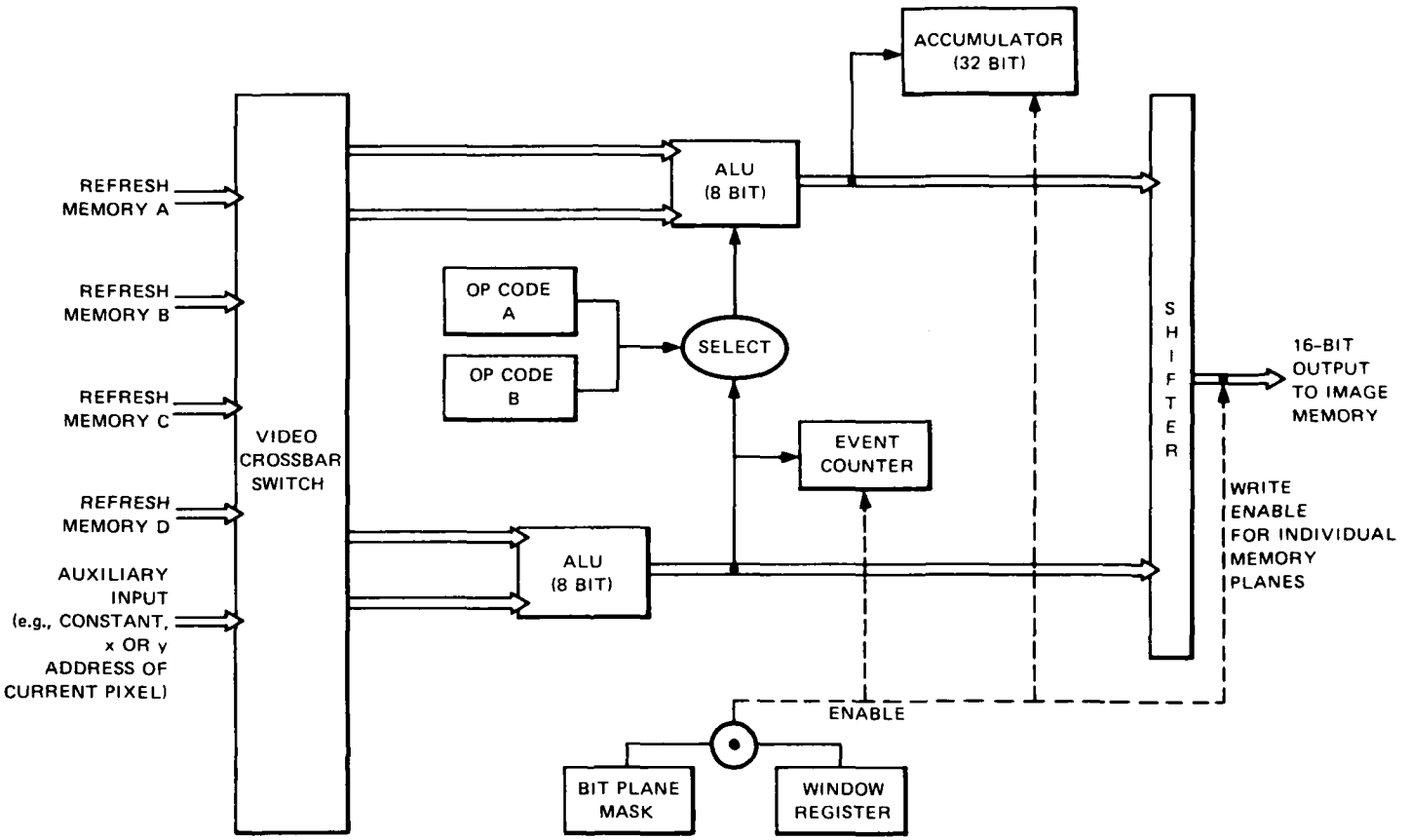


FIGURE 6 SIMPLIFIED DIAGRAM OF DIGITAL VIDEO PROCESSOR

1. Comparison of De Anza and Generic VIP

Table 4 contrasts specific design features of the De Anza processor and the generic VIP described earlier.

Table 4 reveals many surface similarities: both processors consist essentially of a large amount of image memory; image data can be spatially offset and routed through lookup tables or an ALU; processing can be restricted to areas of an image defined by mask overlays and a rectangular window.

However, there are subtle engineering compromises in the De Anza design that complicate programming and, in some cases, make effective algorithm implementations impossible. The most severe limitations are imposed by the small amount of available memory, rigidly configured as four 8-bit channels, the primitive nature of the ALU, and the interlaced scanning raster; the De Anza scans images in a strict top-bottom, left-right raster, processing only even or odd scan lines on alternate passes through the image. These limitations interact perniciously.

Since the De Anza's ALU is limited to 8-bit addition and subtraction, many single-pixel arithmetic operations must be synthesized on multiple passes. Such synthesis can require multiple image arrays for temporary storage. Not only are scarce memory resources strained, but numeric precision, overflow, and sign are difficult to handle properly because of the rigid 8-bit structure of the arrays. As one example, a medium-size convolution operator can easily require 24 bits of significance, tying up all available memory for temporary storage. In such situations, auxiliary images must be stored on disk, which incurs a severe performance penalty.

The ALU's single accumulator makes statistical computations such as histograms exceedingly slow (1 bin/pass); and x,y coordinate values are not available for computation of moments. Moreover, the architecture precludes adding processors specialized for such functions.

Table 4

COMPARISON OF GENERIC VSP AND DE ANZA DESIGNS

	Generic VSP	De Anza IP-5000
Processor Architecture		
	Flexible bus structure connecting memory with general and special-purpose processors	Fixed data flow, routed thru a single processor
Memory Architecture		
	Stack of bit-planes organizable into image channels with variable precision/pixel	Four 8-bit channels
	Noninterlaced raster scan, reversible scan directions	Interlaced raster scan (strict top-bottom, left-right)
	3 x 3 local window accessible	Single pixel accessible
	Zoom and scroll	Zoom and scroll
Processor boxes		
	Lookup tables Multiple 512 x 9-bit tables	Single 256 x 8-bit table/channel
	Function boxes	Constant register
	General-purpose processor with full complement of 32-bit arithmetic, logical, test and compare operations and 256 internal registers	Primitive ALU: 8-bit add/sub/logical, (no sign bit) + shift; 2-way conditional op-code selection
		Statistical accumulator and event counter, readable only by the host processor
	Mask plane and window registers	Mask plane and window registers

Propagation algorithms cannot be implemented reasonably. They are excluded by numerous factors: the need for multiple passes to synthesize complex pixel operations, resulting in exponential growth in computing time; the absence of temporary storage in the ALU, needed to propagate information left to right across an image; the field interlace scan pattern, which makes it hard to propagate information top-bottom down an image; and the inability to reverse scan direction, which excludes algorithms requiring both a forward and backward pass.

Statistics and propagation aside, the IP-5000 is well suited for many single-pixel and local neighborhood operations. Simple functions such as adding or logically combining images in the ALU and scaling images in lookup tables are efficiently implemented. Such operations support a wide variety of tasks; for example, noise reduction (averaging multiple TV frames), real-time photometric camera correction (subtracting a reference image and scaling the result in a lookup table), intensity transformations (e.g., mapping linearly quantized gray values onto a log scale), and multispectral classification (subtracting logarithmic images and classifying the resulting ratios using a lookup table).

Local 3 x 3 binary neighborhood operations (e.g., thinning and thickening) can be implemented in nine passes by converting spatial patterns into an equivalent image, and passing that image through a lookup table, as described for the generic VSP. Local arithmetic operations (e.g., spatial derivatives, median filter) can also be handled as in the generic processor, except for complications that may arise due to the simplicity of the ALU and the limited memory capacity and precision.

Convolution and correlation can be synthesized in multiple passes (one pass/mask element), as described for the generic processor, again subject to memory limitations.

VI EXPERIMENTAL RESULTS

A. Introduction

A set of representative image-processing algorithms were selected for experimental evaluation on the De Anza processor. The algorithms were selected for their cartographic utility and the range of De Anza features they illustrate.

B. Work Station

The work station used in the experiments is shown in Figure 7. The camera and light stage on the left provide an on-line source of imagery, which is displayed on the top black and white monitor. Digitized imagery from De Anza memory is displayed on the lower color monitor, which features a 19-in., high-resolution shadow mask CRT manufactured by Mitsubishi. A joystick and conventional computer terminal complete the station.

C. Single-Pixel Functions

Simple arithmetic and logical functions of a single pixel in one or more images can be performed on the De Anza in one pass. A number of such operators were implemented to gain familiarity with the hardware, including thresholding and image scaling, which are performed using lookup tables, and image addition and subtraction, which are performed in the ALU.

We also implemented a double-precision (16-bit) arithmetic package, using the dual ALU architecture of the De Anza processor. Two images serve as accumulators, storing the high- and low-order bytes, respectively. To illustrate double-precision operations, suppose an 8-bit image was to be added to a 16-bit accumulated sum. The lower ALU

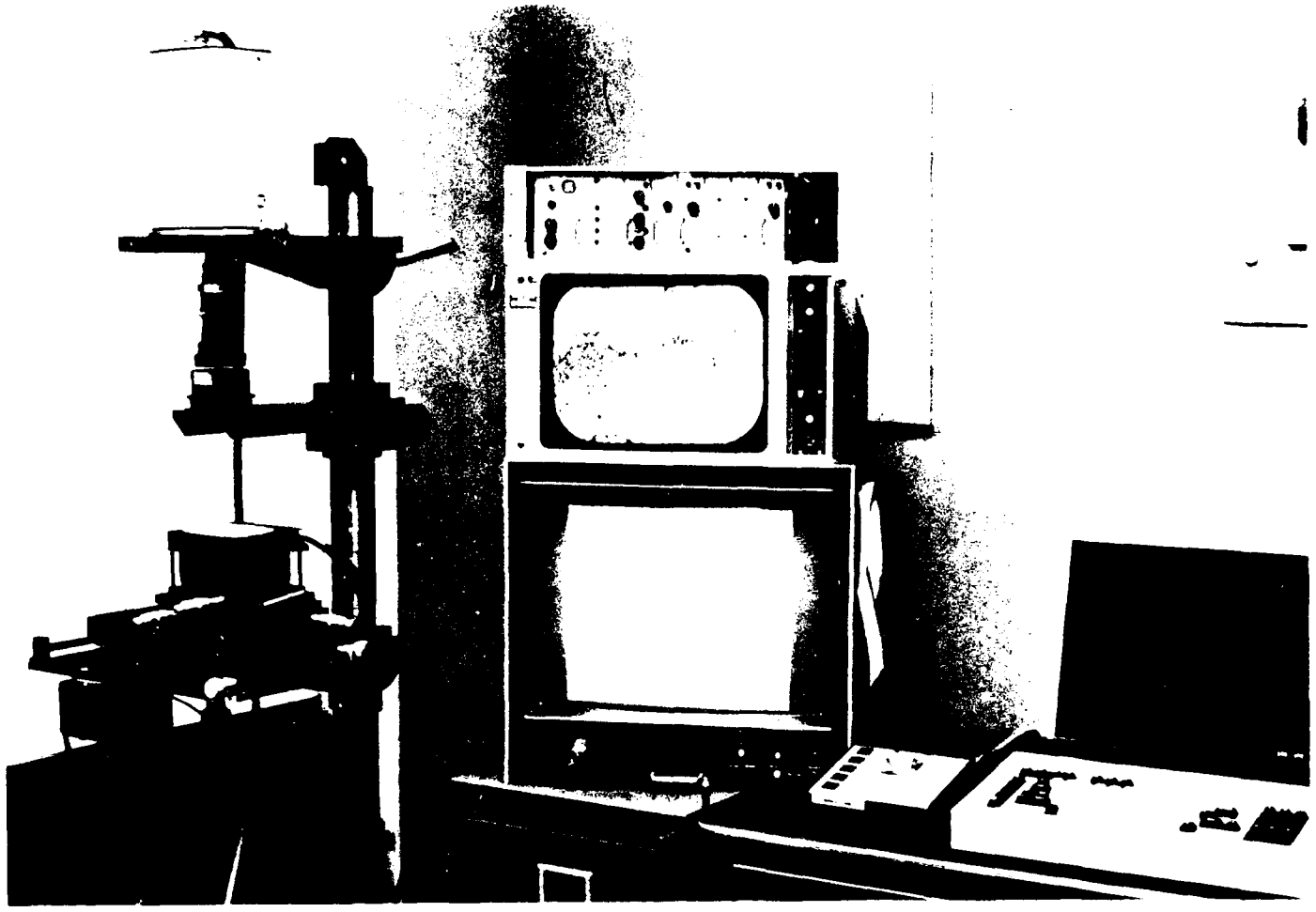


FIGURE 7 DE ANZA IMAGE-PROCESSING WORK STATION

would form the sum of the addend image and the low-order byte of the accumulator image. The upper accumulator would sum the high-order accumulator byte with any carry generated by the lower ALU. Double- and even triple-precision sums, formed in an additional pass, are used extensively in convolution and correlation operations described later.

D. Real-Time Image Enhancement

The single-pixel operations described in the preceding section were used to improve the image quality obtainable from a TV camera by reducing noise, increasing gray-level resolution, and compensating for photometric nonuniformities.

Noise was reduced by summing successive frames from the 6-bit camera digitizer into a 16-bit (double-precision) accumulator image. Averaging n frames had the anticipated effect of reducing the standard deviation of intensity over a blank field by a factor of $1/\sqrt{n}$.

Gray-level resolution was increased from 6 bits to 8 bits, using a feature of the digitizer that allows comparator levels to be offset by a fraction (1/4, 1/2, and 3/4) of the least significant bit under program control. The sum of four frames at offsets of 0, 1/4, 1/2, and 3/4, respectively, increases the effective resolution of the 6-bit digitizer to 8 bits. Typically, 64 frames are summed, 16 at each offset, to obtain a low-noise, 8-bit image.

The principal photometric nonuniformity in vidicons is a systematic shading across the image due to scan nonlinearities. This artifact is clearly evident in Figure 8, which was obtained by scrolling an image array (with wraparound) so that the left and right edges of a test pattern are seen in juxtaposition. To minimize shading and other photometric nonuniformities, a noise-reduced image array x is subjected to the transformation:

$$I = \frac{X-B}{W-B} \times 256$$

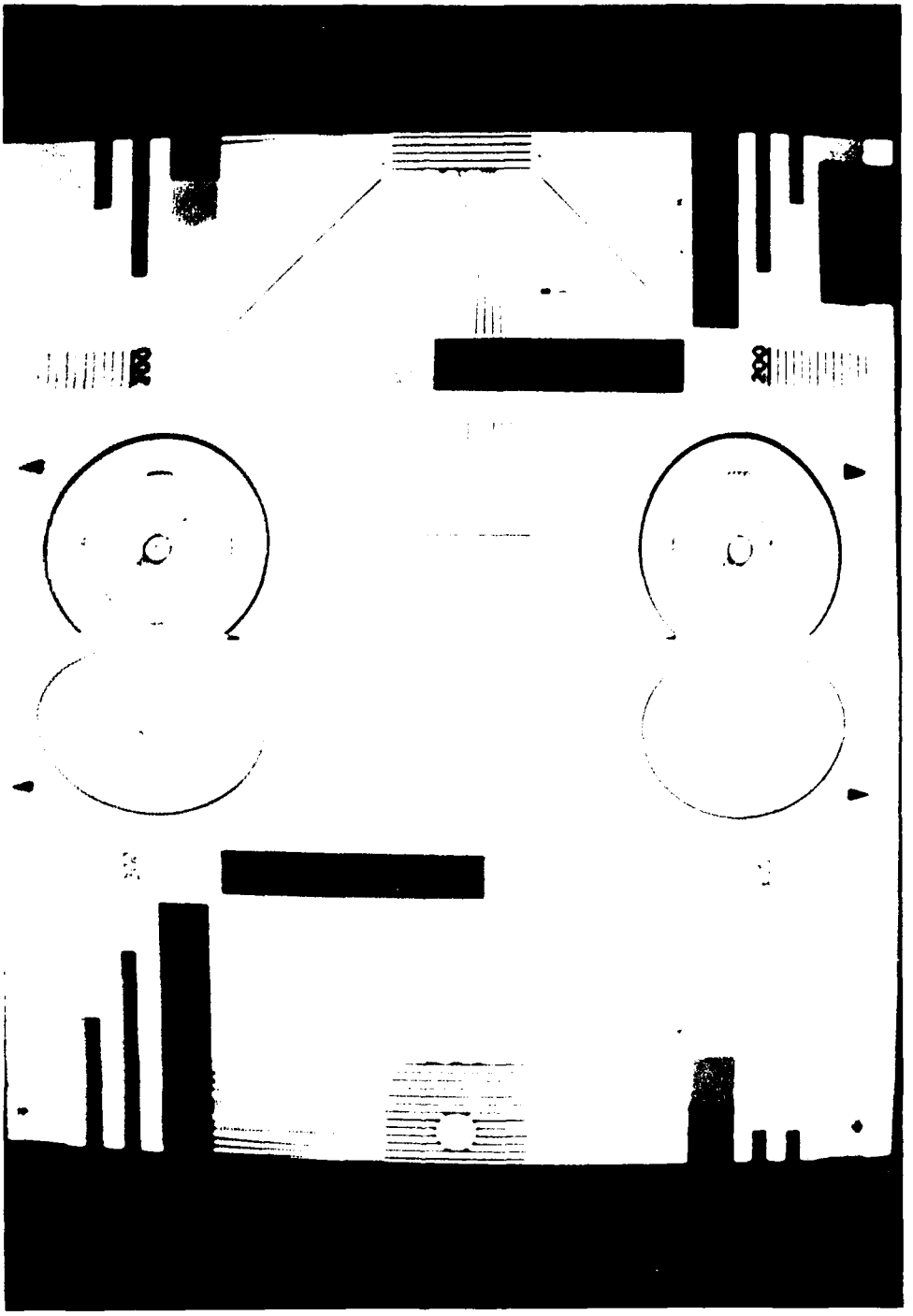


FIGURE 8 SHADING DISTORTION IN VIDICON IMAGERY

where I is the corrected image array, B is an image array of a blank field (obtained with the lens cap on the camera), and W is an image of a uniform white field. Scaling by 256 restores contrast.

This photometric correction was implemented using both the De Anza ALU and the PDP-11 host processor. We first digitized low-noise, 8-bit images of a blank field (B) and a white field (W), as previously described. Difference images (X-B) and (W-B) were then computed in the De Anza ALU and subsequently divided in the PDP-11 ALU, which accessed De Anza memory via the memory-mapping hardware. The resulting ratio image was then scaled (by 256) by passing it through a lookup table. The division could, in principle, have been performed in the De Anza ALU. However, it would have required at least eight passes through the image and would have involved a substantial amount of programming effort to maintain the required arithmetic precision. Figure 9 shows the improvement resulting from applying the photometric correction to the test pattern image of Figure 8.

E. Unsharp Masking

Unsharp masking was implemented as an example of hybrid optical-digital processing. An image was first read into a De Anza image array. The camera was then defocused to obtain an optically low-pass filtered image. This low-passed image was subtracted in the ALU from the original to produce a high-frequency, enhanced image.

F. Edge Detection

Simple vertical and horizontal edge detectors were implemented, each requiring two passes. In the first pass the image is copied into another array through the ALU. On the second pass, the copy is scrolled horizontally (or vertically) by one or two pixels and subtracted in the ALU from the original.

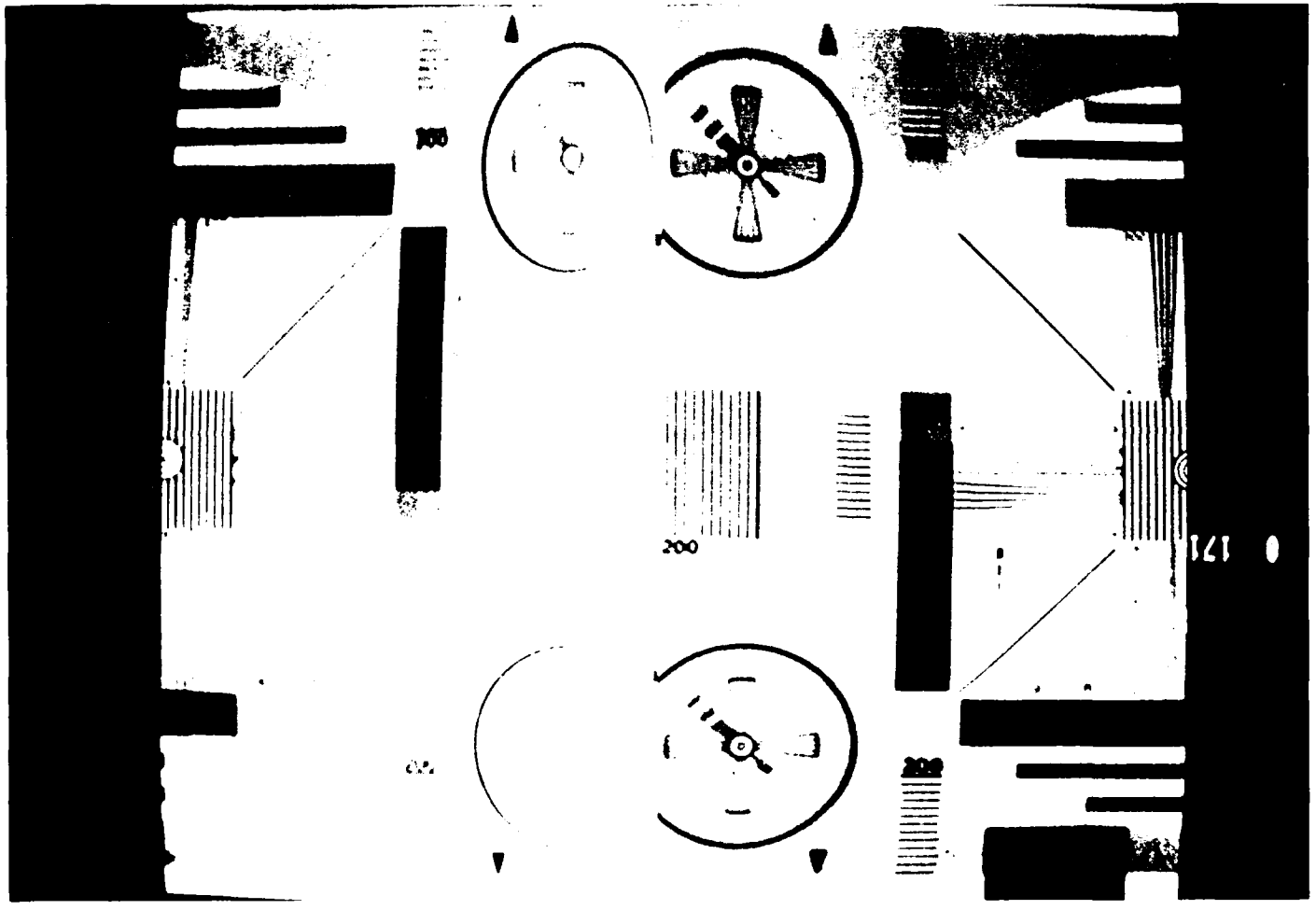


FIGURE 9 PHOTOMETRICALLY CORRECTED VIDICON IMAGE

G. Convolution

The convolution of an image with an nxn mask can be implemented in n^2 passes. In each pass, the input image is spatially shifted and multiplied in a lookup table by the weight at the corresponding mask position. The resulting product is then summed (or subtracted, depending on sign of the weight) into a 16-bit accumulator array. If additional precision is required, the convolution algorithm can be extended to 16-bit weights and 24-bit accumulation; two passes per mask point are required (see Figure 10). First, a lookup table is initialized to multiply the input image by the low-order 8 bits of the current weight. The result is summed in the low-order 16 bits of a 3-channel accumulator image. In the second pass, the lookup table is used to multiply by the high-order 8 bits of the current weight and the results summed with the high-order 16 bits of the accumulator. The high-order channel of the 24-bit sum will nominally contain the most significant information. However, in many cases it will contain all zeros or all ones. In such cases, a significant display can be obtained by performing a logical left shift in the ALU shift register, on the most significant 16 bits of the accumulator array. The amount of shift is the smallest needed to obtain an 8-bit image where some significant percentage of pixels have different values for their high-order bit.

H. Correlation

Unnormalized correlation for matched filtering was implemented using the same approach as convolution; i.e., shifting the input image, scaling it in a lookup table by the weight at that corresponding mask position, and summing into an accumulator image. To demonstrate matched filtering, a PDP-11 program was written that allows a small square mask to be interactively positioned in an image using the joystick. The De Anza processor then performs a correlation using as weights the pixel values in the mask.

In Figure 11 a mask has been positioned on a diagonal line sloping downward left to right. Figure 12 is the result of correlating the test

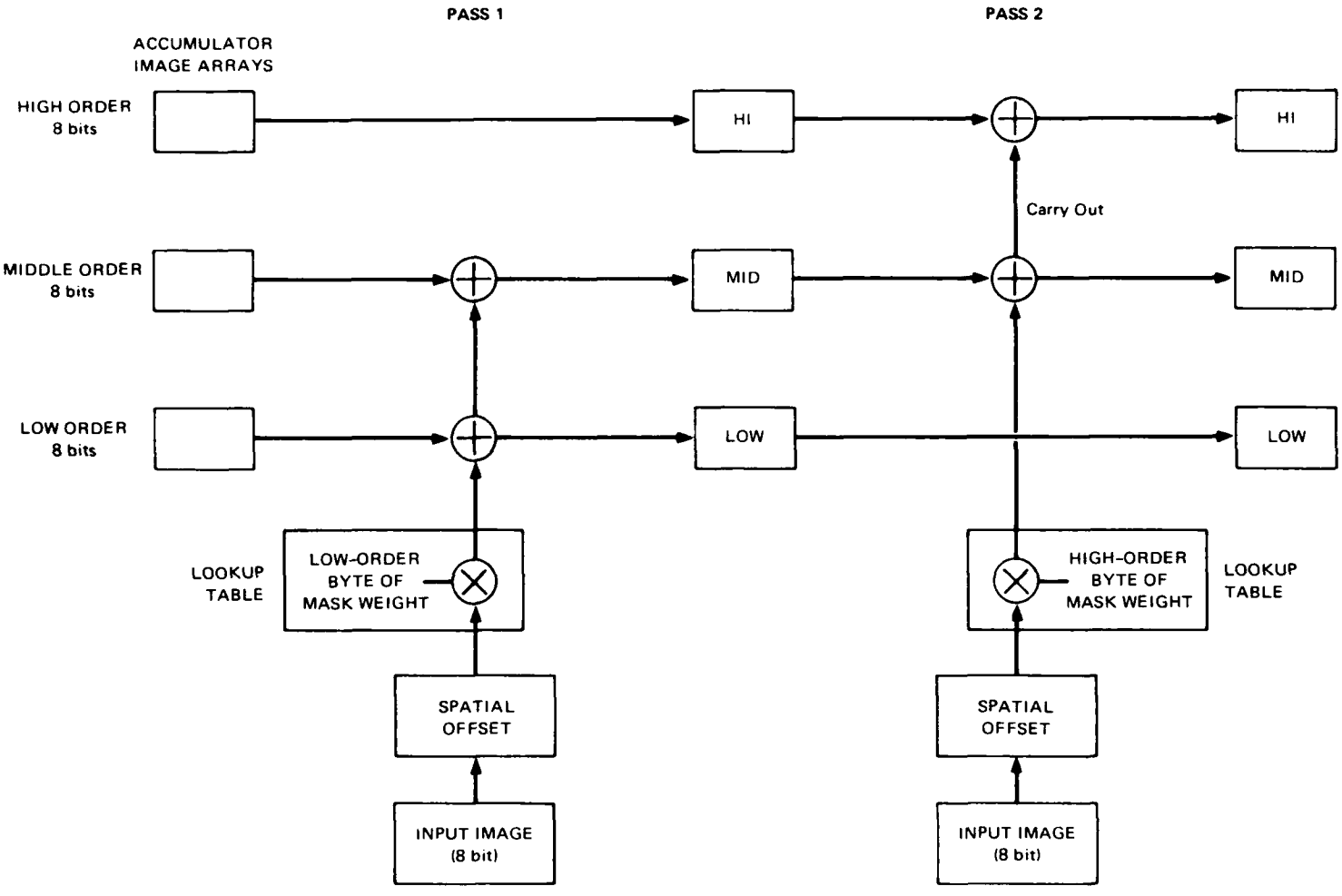


FIGURE 10 DATA FLOW FOR TWENTY-FOUR-BIT CONVOLUTION (16-bit MASK, 8-bit Image)

These two passes are repeated, with an appropriate spatial offset, for each mask element.

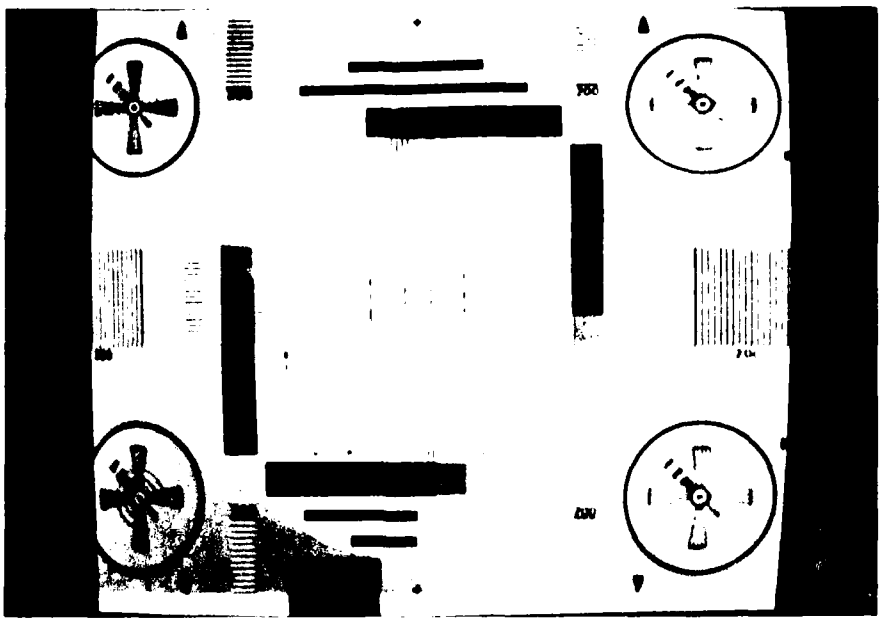


FIGURE 11 SMALL WHITE BOX (ON UPPER LEFT DIAGONAL) DENOTES CORRELATION MASK

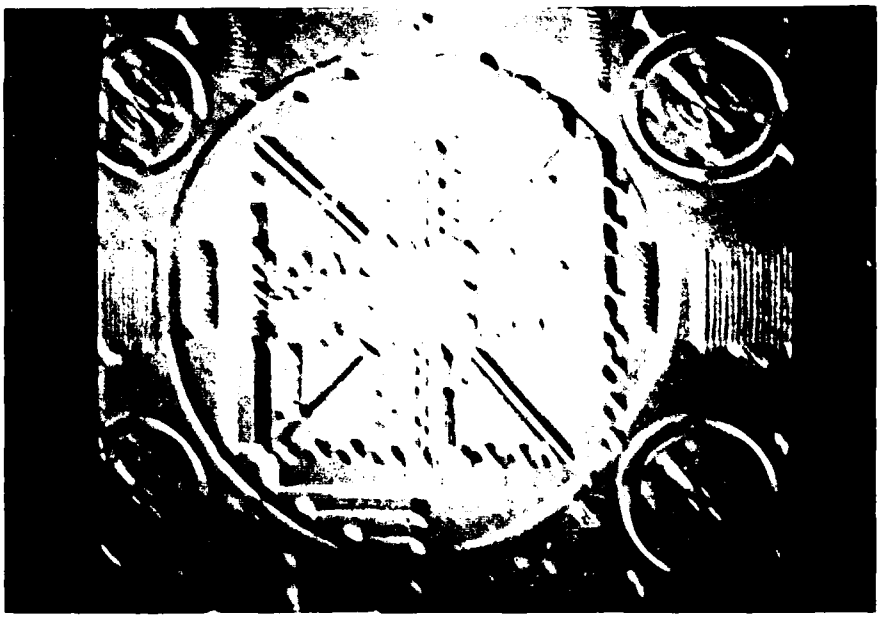


FIGURE 12 RESULT OF CORRELATING FIGURE 11 WITH MASK REGION DEFINED BY SMALL WHITE BOX (BRIGHTNESS DENOTES DEGREE OF CORRELATION)

pattern image with this diagonal line mask, with degree of correlation represented by brightness. Similarly, Figure 14 is the result of correlating an aerial image with a mask array taken from the plowed field (Figure 13).

The most sophisticated application of correlation to date on the De Anza was a fast implementation of David Marr's theory of edge detection [22]. In Marr's theory, edges correspond to zero crossings in the band-pass function that results when an image is convolved with a mask whose values represent the difference of two concentric, two-dimensional Gaussians, with widths in the ratio 1:1.7. (This mask is a center surround operator resembling a Laplacian.) Various size masks are used, the smallest of which contains nearly 1,000 pixels. Masks of this size require maintaining a full 24 significant bits of intermediate results during convolution. Results of zero crossing edge detection were identical in appearance with those in Marr's paper.

I. Interactive Overlay Generation

An important function in military geographic intelligence is the generation of overlays that delimit specific areas of an image, such as areas of vegetation, trees, or concrete. The creation of such overlays can be partially automated, using the interactive image display and high-speed processing capabilities found in the De Anza. Figure 16, for example, is a shadow overlay, produced by passing the image in Figure 15 through a lookup table initialized to emphasize dark areas. Figure 17 is a vegetation overlay, produced by convolving Figure 15 with a vertical edge mask. These overlays can now be thresholded and logically ANDed to produce a new overlay of dark areas with high-frequency content that are characteristic of trees.

The quality of the above overlays could possibly be improved through use of additional spatial and textured operators. The processing speed of the De Anza, coupled with its immediate visual feedback, encourages an interactive approach to overlay generation, in which empirically determined combinations of image operators are used to delimit particular scene features.

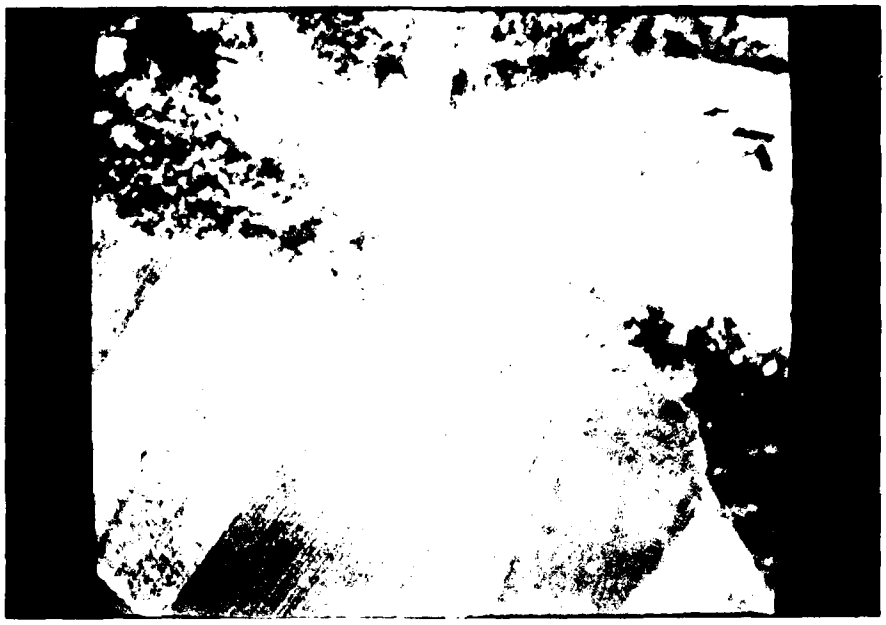


FIGURE 13 AERIAL VIEW OF PLOWED FIELD

FIGURE 14 RESULT OF CORRELATION WITH A MASK TAKEN FROM FIGURE 13 (THE UPPER FIELD IS DENOTED BY THE SMALL WHITE BOX SUPERIMPOSED ON THIS FIGURE)

FIGURE 15 A TEST IMAGE TO ILLUSTRATE INTERACTIVE OVERLAY GENERATION

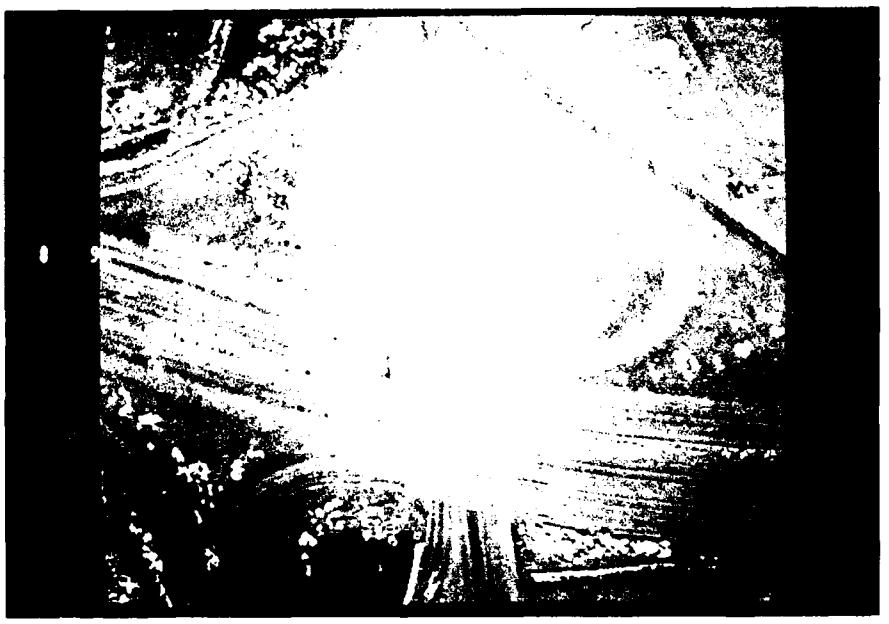


FIGURE 16 A SHADOW OVERLAY PRODUCED BY THRESHOLDING IN FIGURE 15 TO EMPHASIZE DARK AREAS

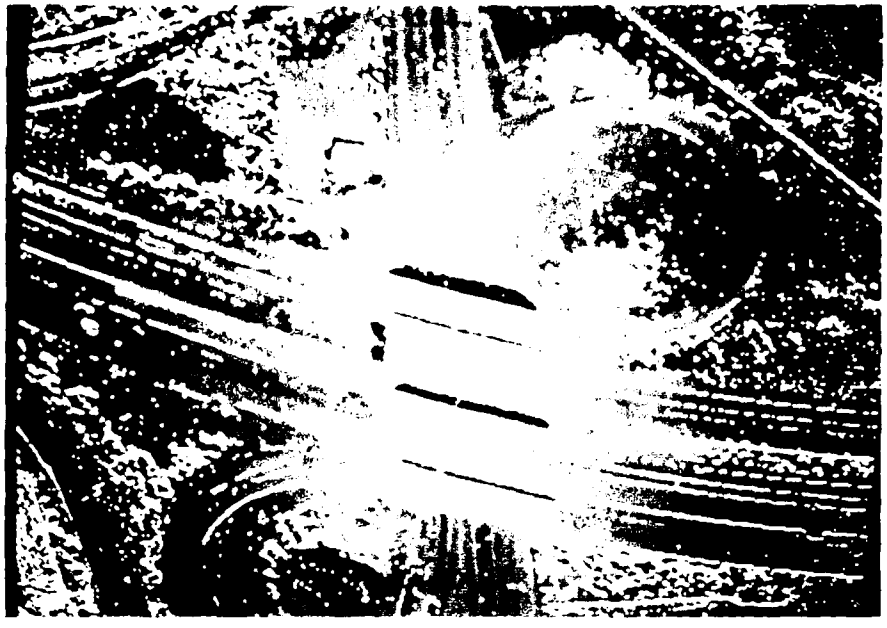


FIGURE 17 A VEGETATION OVERLAY PRODUCED BY CONVOLVING FIGURE 15 WITH AN EDGE MASK

VII DISCUSSION

VSPs provide a cost-effective means for performing any image-processing function that can be implemented with local parallel operations. Essentially, this class includes all commonly used low-level algorithms (with the exception of geometric transforms and sequential line trackers), as well as the most computationally intensive parts of many higher-level image-understanding algorithms.

A VSP simulates a parallel-array processor at a fraction of the cost by streaming image data at video rates through a single processing unit. Throughputs comparable to those of a peripheral parallel processor such as Staran (typically two to three orders of magnitude faster than a general-purpose computer) are achieved by keeping the entire image in high-speed memory, thus avoiding I/O bottlenecks.

VSP systems evolved from image displays by adding limited processing capabilities to existing memory resources. The De Anza IP-5000 is one of the most advanced such systems available today. It can synthesize most single-pixel and local neighborhood operations, with the notable exception of certain global statistics (e.g., histograms and moments) and propagation algorithms (e.g., distance transforms). However, many operations require an excessive number of passes through the image and excessive temporary storage to implement. Moreover, programming is done at the hardware level and is very difficult.

The existing De Anza design can be refined to ameliorate most of the problems associated with storage requirements and programming difficulty. First, the primitive ALU should be replaced with a full 16-bit processor that can perform 8-bit signed operations, including multiplication and division in a single-frame time. Accumulators and temporary registers must be added for statistics calculations; or, alternatively, a separate statistics processor should be installed.

Additional image memory is needed to accommodate temporary results that require high precision. With these improvements, many multipass operations could be streamlined to one or a few passes. Equally important, programming could be elevated from concerns about sign, overflow, and precision to a level concerned with manipulation of image arrays. However, to handle propagation algorithms, the IP-5000 would have to be substantially redesigned to more closely resemble the generic VSP.

Propagation algorithms would require a bus structure to accommodate specialized processing units (e.g., adder trees) capable of processing all pixels in a local neighborhood in a one-pixel interval. The memory architecture would also have to be modified to provide simultaneous access to a 3 x 3 window at each location, to allow raster scans in arbitrary directions, and to eliminate the field interlace that sequentially processes alternate lines.

A generic VSP incorporating the above refinements could be built by many of the leading display manufacturers. Technological feasibility has, indeed, already been demonstrated by the TOSPICS System manufactured by Toshiba. However, TOSPICS is priced out of the market because it is based on obsolete memory technology.

At this point, it is appropriate to address a fundamental limitation inherent in the design of VSPs: the strict video raster format of memory access that subordinates processing to display refresh and so greatly complicates the implementation of functions such as propagation, geometric transforms, and sequential edge trackers.

The raster format has two drawbacks. First, it requires that all processing be completed for each pixel location within a strict 100-nsec window. This constraint not only leads to complicated and expensive high-speed processing units for propagation algorithms that must be completed in one pass, it also tends to squander memory resources. Whenever a computation cannot be completed in one pass, temporary results must be stored for every pixel. This requires a whole image stack, frequently one with many planes to maintain precision. By

contrast, if the scan were slowed so that processing could be completed at each point before moving on, then temporary results would never have to be stored for more than one pixel location at a time. The second drawback with raster processing is the strictly sequential ordering of memory access. This effectively precludes algorithms such as geometric image transforms that require random access to image arrays.

One way of liberating complex processing from the constraints of video refresh without abandoning the many good features of VSPs is to exploit the De Anza's dual-ported memory. The present PDP-11/34 host computer could be replaced by a powerful general-purpose processor such as a DEC VAX or Motorola M68000. Both these computers have large (32-bit) address spaces that allow direct access to De Anza's bulk image store. They are also user-microprogrammable, so that array access computations can be handled efficiently in micro code. Any algorithm involving spatial context, nonuniform memory access, or excessive temporary storage could then be implemented on the host. Meanwhile, the image memories could still be accessed through the second part in raster mode for both image display and simple stream processing.

Looking further ahead, one can ask how emerging developments in solid-state technology are likely to impact image processor design. The size and geometric regularity of image rasters and the uniformity and locality of most operations make image processing an ideal candidate for VLSI implementation. For the first time, massive parallel-array processors in the style of Staran and Illiac IV may be practical on a sufficient scale to process an entire image at once. The beginnings of such developments are already evident in ARPA-sponsored research on smart sensors [23] and emerging designs for cellular image-processing architectures. However, general-purpose systems suitable for image-processing research are still far off.

VIII CONCLUSION

VSPs provide the most cost-effective means for implementing local image-processing operations with today's technology. Currently available systems, however, are still primitive; and substantial opportunities remain for further development. One promising approach is a hybrid design, based on a dual-ported memory architecture, that integrates the generic VSP outlined in this report with a powerful general-purpose processor, such as a VAX or M68000.

REFERENCES

1. D. Rohrbacker and J. L. Potter, "Image Processing with the Staran Parallel Computer," Computer, Vol. 10, No. 8, pp. 54-60 (August 1977).

2. L. A. Gambino and B. L. Schrock, "An Experimental Digital Interactive Facility," Computer, Vol. 10, No. 8, pp. 22-29 (August 1977).

3. J. Adams and R. Wallis, "New Concepts in Display Technology," Computer, Vol. 10, No. 8, pp. 61-69 (August 1977).

4. A. Rosenfeld and A. C. Kak, Digital Picture Processing, W. Rheinboldt, ed. (Academic Press, New York, New York, 1976).

5. H. G. Barrow et al., "Parametric Correspondence and Chamfer Matching: Two New Techniques for Image Matching," Proc. 5th International Joint Conference on Artificial Intelligence, M.I.T., Cambridge, Massachusetts (August 1977).

6. M. A. Fischler and P. Barrett, "An Iconic Transform for Sketch Completion and Shape Abstraction," Technical Note 195, Artificial Intelligence Center, SRI International, Menlo Park, California (October 1979); to appear in Computer Graphics and Image Processing (1980).

7. J. Munson, internal memo, Artificial Intelligence Center, Stanford Research Institute, Menlo Park, California, December 1973.

8. A. Rosenfeld and J. L. Pflatz, "Distance Functions on Digital Pictures," Pattern Recognition, Vol. 1, No. 1, pp. 33-62 (July 1968).

9. L. H. Quam, "Euclidean Distance Transformations of Binary Images" (paper in preparation).

10. O. Firschein and M. A. Fischler, "Association Algorithms for Digital Imagery," Conference Record of the Twelfth Annual Asilomar Conference on Circuits, Systems, and Computers, IEEE Catalog No. 78CH1369-8C/CAS/CS (November 1978).

11. D. H. Bass, "Using The Video Lookup Table for Reflectivity Calculations: Specific Techniques and Graphic Results," TR51, Computer Science Dept., University of Rochester, Rochester, New York (September 1979).

12. J. M. Tenenbaum, "Accommodation in Computer Vision," AIM-134, Computer Science Department, Stanford University, Stanford, California (1970).

13. C. L. Fennema and W. B. Thompson, "Velocity Determination in Scenes Containing Several Moving Objects," Computer Graphics and Image Processing, Vol. 9, No. 4, pp. 301-315 (April 1979).

14. J. P. Tippett et al., Optical and Electro-Optical Information Processing (M.I.T. Press, Cambridge, Massachusetts, 1965).

15. A. D. Gara, "Optical Computing for Image Processing," in Computer Vision and Sensor-Based Robots, G. Dodd and L. Rossol, eds. (Plenum Press, 1979).

16. G. Lendaris and G. Stanley, "Diffraction-Pattern Sampling for Automatic Pattern Recognition," Proc. IEEE, Vol. 58, No. 2 (February 1970).

17. G. J. Agin and T. O. Binford, "Computer Description of Curved Objects," Proc. 3rd International Joint Conference on Artificial Intelligence, Stanford University, Stanford, California (1973).

18. R. Ohlander, K. Price, and R. Reddy, "Picture Segmentation Using a Recursive Region-Splitting Technique," Computer Graphics and Image Processing (in press).

19. T. D. Garvey and J. M. Tenenbaum, "Application of Interactive Scene Analysis Techniques to Cartography," in Proc. 3rd International Joint Conference on Pattern Recognition, p. 213 (IEEE Computer Society Publication No. 76CH1140-3C, November 1976).

20. M. .A. Fischler, J. M. Tenenbaum, and H. C. Wolf, "Detection of Roads and Linear Structures in Low-Resolution Aerial Imagery Using a Multisource Knowledge Integration Technique," Technical Note 200, Artificial Intelligence Center, SRI International, Menlo Park, California (December 1979); to appear in Computer Graphics and Image Processing (1980).

21. H. G. Barrow and J. M. Tenenbaum, "Recovering Intrinsic Scene Characteristics from Images," in Computer Vision Systems, A. Hanson and E. Riseman, eds., pp. 3-26 (Academic Press, New York, New York, 1978).

22. D. Marr and E. Hildreth, "Theory of Edge Detection," AI Memo S18, M.I.T., Cambridge, Massachusetts (1979).

23. Proceedings ARPA Image Understanding Workshops, Lee Baumann, ed. (Science Applications, Inc., Arlington, Virginia).


FILME